Apple Answer Book 2

This is the second volume of the *Apple Answer Book.* Volume 1 was a tremendous success and we hope to continue this trend.

This book is designed to present helpful information that isn't in your reference manuals and explanations of common problems. The information in this volume may duplicate some of the items in volume 1. In this case, the material presented here supercedes the previous item.

The *Answer Book* needs you. If you have some information or a solution you'd like to share with other Apple users, send it to us here in Marketing Applications. We may publish it in a future Answer Book, in the Apple Orchard or in a dealer mailing.

Spread the good word!

Jo Kellner
Marketing Applications

THE ANSWER BOOK VOLUME 2 TABLE OF CONTENTS

SECTION I. Apple II System Software

Applesoft
- User Defined Functions and CHAIN
- HIRES "SCRN" Function for Applesoft
- Literal Input Routine for Applesoft
- Applesoft Array Eraser
- Applesoft Reference Manual Errata
- Applesoft Tutorial Errata

DOS
- The DOS Manual Errata

FORTRAN
- FORTRAN System Requirements
- Unformatted I/O in Apple FORTRAN
- Memory Space for FORTRAN
- FORTRAN Copy-Protection
- Using Library Units in FORTRAN with Pascal 1.1
- Creating FORTRAN Data Files
- FORTRAN: Making the Most of Your Disk Space
- FORTRAN Language Reference Errata

Pascal
- Pascal Compiler Options
- Upper and Lower Case in Pascal 1.1 and Pilot
- Passing Strings To and From Functions
- Pascal Peripheral Slot Assignments
- Nesting Pascal Procedures and Functions
- Pascal Memory Space
- Pascal Immediate Mode
- Pascal Turnkey System
- Pascal External References
- Pascal Inverse Video
- Making a Copy of "BASICS"
- TTLOUT
- Pascal Scan Function
- Rebooting Problems in Pascal
- Chaining in Pascal
- Pascal Booleans
- Pascal TRUNC and ROUND Functions
- Proposed ISO Standards for Pascal
- Pascal Error Messages
- Pascal Run-Time Errors

SECTION I. (continued)

Pascal
Pascal Real Number Format
Using Resident and Noload Compiler Options with Intrinsic Units
The Keypress Function
Pascal Units

Pascal Long Integer Compares - Version 1.0 Only
Converting Strings to Numeric Variables
Pascal Unit for Silentype Procedures
Pascal Operand Formats

External Terminal Setup in Pascal
Pascal Language Reference Manual Errata
Pascal Operating System Manual Errata
ATTACH-BIOS Document for Apple II Pascal 1.1

SECTION II. Apple II Applications Software

Apple Post
Apple Post and DOS 3.3
Apple Post and Printers
Labels and the Left Margin in Apple Post
Zip Codes in Apple Post

Apple Post File Size
Apple Post and the Sample List
FORMAT Statement in Apple Post

Apple Plot
Apple Plot and Visicalc File Converter
Modifying the Apple Plot Program
Qume Graphics Driver for Apple Plot
Apple Plot with Other Graphics Printers

Apple Plot and DOS 3.3
Apple Plot Errors

Portfolio Evaluator and News and Quotes Reporter
Receiving Options with the Portfolio Evaluator
Corrections to Portfolio Evaluator
Portfolio Evaluator and the Micromodem
Dow Jones News and Quotes Reporter

Apple Writer
Apple Writer Modification for Lower Case Display

Elementary, My Dear Apple
Elementary, My Dear Apple Errata

SECTION II. (continued)

Applesoft Tool Kit
Applesoft Tool Kit Manual Errata

Apple 6502 Assembler/Editor
Apple 6502 Assembler/Editor Errata

The Shell Games
The Shell Games Manual Errata

Tax Planner
Tax Planner Manual Errata

SECTION III. Apple II Hardware

Apple II Reference Manual Errata

SECTION IV. Apple II Peripherals

Tabbing with Apple Peripherals
Initializing Apple Peripherals with POKEs
Carriage Return Delay
High Speed Serial Interface Manual Errata

Addendum to the High Speed Serial Interface Manual Errata
Communications Interface Manual Errata
Parallel Printer Interface Manual Errata
Centronics Interface Card

Silentype Manual Errata
Graphics Tablet Manual Errata

SECTION V. Apple /// System Software

Apple /// Business Basic
Printing from Apple /// Business Basic
Apple /// System Configuration Program
Printing From Apple /// Emulation Mode

SECTION VI. Apple /// Applications Software

Printing from Visicalc ///
Visicalc /// Column Insertions

SECTION VII. Apple /// Hardware

Video on the Apple ///
IRQ on the Apple /// in Emulation Mode
Apple /// ACIA
Apple /// Diagnostic Display

Map of the Apple /// Memory Board
Apple /// Emulation to Apple II Hardware Comparison

SECTION VIII. Apple /// Peripherals

Modem Eliminator for Apple ///
Apple /// Joystick Interface

SECTION IX. Vendor Lists

SECTION X. Bibliography

SECTION XI. International Apple Core Member User Groups

Apple II
System Software

USER DEFINED FUNCTIONS AND CHAIN

User defined functions in Applesoft may cause problems if CHAIN is used. When a DEF FN statement is encountered in Applesoft there is an entry made in the simple variable table that points to the rest of the function in the text of the program. Strange and perhaps fatal things can happen if you use a function defined in the previous program without having the same image of the function at the same memory locations.

The easy way around this problem is not to use defined functions. If you need defined functions then put all the definitions in the front of ALL chained modules. For example:

Program 1:

```
10 DEF FN A(X)=X*X
20 PRINT FN A(2)
30 PRINT CHR$(4);"BLOAD CHAIN,A520"
40 CALL 520"PROGRAM 2"
```

Program 2:

```
10 DEF FN A(X)=X*X
20 PRINT FN A(3)
30 END
```

HIRES "SCRN" FUNCTION FOR APPLESOFT

```
1000  REM    HIRES SCRN FUNCTION DEMO
1010  REM
1020  REM LOAD IN THE BINARY STUFF
1030  REM
1050  FOR J = 768 TO 806: READ K: POKE J,K: NEXT J
1060  DATA 32,227,223,133,133
1070  DATA 132,134,169,208,32
1080  DATA 192,222,165,18,72
1090  DATA 165,17,72,32,185
1100  DATA 246,32,17,244,165
1110  DATA 48,49,38,240,2
1120  DATA 169,1,168,32,1
1130  DATA 227,76,91,218
1140  POKE 1013,76: POKE 1014,0: POKE 1015,3
1150  REM ----------------------
1160  REM    DRAW SOMETHING
1170  REM ----------------------
1180  LET HO = 120:VO = 60
1190  HGR
1200  HCOLOR= 3
1210  HPLOT HO,VO
1220  FOR J = 1 TO 10
1230  HPLOT TO RND (9) * 40 + HO, RND (9) * 40 + VO
1240  NEXT
1250  HOME
1260  VTAB 22
1270  FOR J = 0 TO 3000: NEXT
1280  REM ----------------------
1290  REM  CONVERT IT TO LORES
1300  REM ----------------------
1310  GR
1320  COLOR= 2: FOR V = 0 TO 39: HLIN 0,39 AT V: NEXT
1330  FOR V = 0 TO 39
1340  FOR H = 0 TO 39
1350  COLOR= 12: PLOT H,V
1360  REM ----------------------
1370  REM     THIS IS IT!!
1380  REM THE SYNTAX IS: &A=B,C   WHERE
1400  REM A WILL GET THE 1 OR 0 - B,C ARE
1410  REM THE HIRES COORIDINATES AS IN HPLOT
1440  REM ----------------------
1450  & A = H + HO,V + VO
1460  COLOR= A * 15
1470  PLOT H,V
1480  NEXT
1490  NEXT
```

LITERAL INPUT ROUTINE FOR APPLESOFT

This routine allows anything you can type to be entered into a string.

```
100  LET IN$ = "X"
110  TEXT : HOME
120  REM ** THE FIRST VARIABLE DEFINED MUST BE A STRING
130  REM ** THIS STRING WILL RECEIVE INPUT FROM THE CALL
140  REM ** THIS POKES THE INPUT SIMULATOR ROUTINE INTO MEMORY
200  FOR J = 768 TO 790
210  READ I : POKE I,J
220  NEXT J
230  DATA 162,0,32,117,253,160,2
240  DATA 138,145,105,200,169,0
250  DATA 145,105,200,169,2,145
260  DATA 105,76,57,213
300  REM ** NOW TO USE IT!
310  PRINT "TYPE IN ANY CHARACTERS YOU WISH:" : PRINT
320  CALL 768:IN$ = MID$ (IN$,1)
330  REM ** THIS IS AN "INPUT IN$" BUT IGNORES "," AND ":"
400  PRINT : PRINT "AND HERE'S WHAT YOU TYPED IN:"
420  PRINT : PRINT IN$
430  PRINT : PRINT "NOTE THAT EVEN QUOTES, COMMAS AND"
450  PRINT "COLONS GET THROUGH UNSCATHED."
460  PRINT : PRINT "NOW LET'S WRITE IT TO THE DISK"
470  PRINT CHR$ (4);"OPEN TEMP"
480  PRINT CHR$ (4);"WRITE TEMP"
490  PRINT IN$
500  PRINT CHR$ (4);"CLOSE"
510  PRINT : PRINT "AND READ IT BACK IN..."
520  LET IN$ = " "
530  PRINT CHR$ (4);"OPEN TEMP"
540  PRINT CHR$ (4);"READ TEMP"
550  CALL 768:IN$ = MID$ (IN$,1)
560  PRINT CHR$ (4);"CLOSE"
570  PRINT : PRINT IN$
580  PRINT : PRINT "TA-DAA!!": END
```

APPLESOFT ARRAY ERASER

```
100  HOME : VTAB 5
120  PRINT TAB (14);"ARRAY ERASER"
130  GOSUB 10000 : GOSUB 1050
150  PRINT "HERE ARE THE ARRAYS"
160  LIST 180 : GOSUB 1000
180  DIM A(100),B(100)
190  PRINT "AND HERE'S WHAT WE PUT IN THEM"
200  LIST 220 - 230
210  GOSUB 1000
220  LET A(100) = 100 : LET B(100) = 100
240  PRINT "OK, LET'S PRINT THEM OUT"
250  LIST 260 - 270
260  PRINT A(100) : PRINT B(100)
280  GOSUB 1000
290  PRINT "NOW TO ERASE ARRAY 'A'"
300  LIST 310
310  CALL 768,A : GOSUB 1000
330  PRINT "OK, NOW LET'S PRINT B(100) TO SHOW"
340  PRINT : PRINT "THAT IT'S STILL THERE"
350  LIST 360
360  PRINT B(100) : GOSUB 1000
380  PRINT "NOW LET'S TRY TO PRINT A(100).  THE"
390  PRINT : PRINT "ERROR WE GET PROVES THAT THE"
400  PRINT : PRINT "ARRAY IS GONE."
410  LIST 420
420  PRINT A(100)
430  END
1000  REM  'PRESS ANY...' ROUTINE
1010  VTAB 23 : PRINT TAB (9);"PRESS ANY KEY FOR MORE"
1030  HTAB 20 : GET A$
1050  VTAB 10 : HTAB 1
1070  CALL - 958 : REM CLEAR SCREEN
1080  RETURN
10000  REM THE 'ERASE' POKER
10010  FOR J = 768 TO 823: READ K: POKE J,K: NEXT J
10020  RETURN
10030  DATA 32,177,0,32,217,247
10040  DATA 24,160,2,165,155
10050  DATA 133,66,113,155,133
10060  DATA 60,200,165,156,133
10070  DATA 67,113,155,133,61
10080  DATA 136,56,165,109,133
10090  DATA 62,241,155,133,109
10100  DATA 200,165,110,133,63
10110  DATA 241,155,133,110,160
10120  DATA 0,32,44,254,32,163
10130  DATA 217,76,152,217
```

APPLESOFT REFERENCE MANUAL ERRATA
030-0013-03

Page 7
Table 2, the backspace key (<-) and the forward copy key (->) are reversed.
-> $95 $95 $95 $95
<- $88 $88 $88 $88

Page 15
Line 90 should read
```
  90 DIM A(8): REM DIMENSION ARRAY WITH MAX. 9 ELEMENTS
```

Page 15
Line 180 should read
```
  180 IF A(I) <= A(I+1) THEN GOTO 240
```

Page 21
The line about mid-page that starts
```
  C$ = RIGHT$(B$,3) +
```
should start
```
  C$ = RIGHT$(B$,4) +
```

Page 23
Line 180 should read
```
  180 I = I+1: IF I < 15 THEN GOTO 130
```

Page 51
If TAB(X) is the last item in a PRINT statement Applesoft will act as if there is a semi-colon after it.

Page 52
If SPC(X) is the last item in a PRINT statement Applesoft will act as if there is a semi-colon after it.

Page 53
FRE returns the amount of memory ... Applesoft sometimes stores duplicate strings separately.

Page 66
Similarly, a response... If an ASCII NUL character, CONTROL-SHIFT-P, is included in a line of input, the input will be truncated at the character before the NUL. A NUL as the first character in response to INPUT A$ will result in a null string. A NUL as the first character in response to INPUT A will result in ?REENTER.

Page 66-67
A line of input longer than 255 characters will be cancelled by the monitor and the user is left in the input statement. A line greater than 239 but less than 255 will be trwncated to 239 characters.

Page 81
ONERR GOTO, ´When an error occurs´ should read ´After executing an ONERR GOTO command, when an error occurs´.

Page 118
´1) Use multiple statements´ states that the maximum line number is 65529 when the actual maximum line number is 63999.

Page 131
Second paragraph, line 7, replace ´POKE 22,W´ with ´POKE 33,W´.

Page 135
Replace the top part with

260 POKE -16296,0
Clear game control "annunicator" output #0 (Game I/O connector, pin 15) to TTL low (0.3 vlots). This is the "off" condition: maximum current 8 miliamperes.

270 POKE -16295,0
Set game control output #0 to TTL high (3.5 volts). This is the "on" condition: maximum current 0.4 miliamperes.

280 POKE -16294,0
Clear game control "annunicator" output #1 (Game I/O connector, pin 14) to TTL low (0.3 volts). This is the "off" condition: maximum current 8 miliamperes.

290 POKE -16293,0
Set game control output #1 to TTL high (3.5 volts). This is the "on" condition: maximum current 0.4 miliamperes.

300 POKE -16292,0
Clear game control "annunicator" output #2 (Game I/O connector, pin 13) to TTL low (0.3 volts). This is the "off" condition: maximum current 8 miliamperes.

310 POKE -16291,0
Set game control output #2 to TTL high (3.5 volts). This is the "on" condition: maximum current 0.4 miliamperes.

320 POKE -16290,0
Clear game control "annunicator" output #3 (Game I/O connector, pin 12) to TTL low (0.3 volts). This is the "off" condition: maximum current 8 miliamperes.

330 POKE -16289,0
Set game control output #3 to TTL high (3.5 volts). This is the "on" condition: maximum current 0.4 miliamperes.

Page 137
The table for string pointers is wrong.

STRING POINTERS

NAME	(pos)	1st byte
	(neg)	2nd byte
length		1 byte
address		low byte
address		high byte
	0	
	0	

STRING POINTERS

NAME	(pos)	1st byte
	(neg)	2nd byte

APPLESOFT TUTORIAL ERRATA
030-0044-00

Page 23-24
There are only five different elementary arithmetic operations: "+" (Add), "-" (Subtract), "*" (Multiply), "/" (Divide), "^" (Exponentiation).

Page 59
Line 230 at the bottom of the page should read

```
230 IF N < = 10 THEN GOTO 210
```

Page 63

```
270 Y = Y / 7
```

should read

```
270 X = X / 7
```

Page 71
Lines 650 and 660 are reversed. Swap them and the program will work.

Page 79
The two example line 780s are incorrect. The word 'backround' will be parsed as 'BACK GR OUND' because GR is a key word. Try using 'BACKDROP' instead.

Page 85
Line 220 has an extra ')'. It should be:

```
220 COLOR= INT (16 * RND(1))
```

Page 95
The program will give incorrect values for the Y variable unless some time delay is installed. Add this line:

```
1005 FOR J = 1 TO 10: NEXT J
```

Page 109

```
340 TEMP = GLASS(WINE):GLASS(WINE) = GLASS(MILK):GLASSMILK = TEMP
```

should be

```
340 TEMP = GLASS(WINE):GLASS(WINE) = GLASS(MILK):GLASS(MILK) = TEMP
```

Page 130
Clear the entire screen should be "Press <ESC> then <SHIFT> and <P>" instead of "<CONTROL> and <P>".

Page 148-149
The green keys should have slashed zeros to differentiate then from capital O.

THE DOS MANUAL ERRATA
030-0115-00

Page 178
The first paragraph mentions LOADAPA which it shouldn't, because LOADAPA is a part of the DOS Tool Kit.

Page 182
We need to stress that the BASICS is a Pascal diskette and it must boot in slot 6. We also should talk about the BOOT13 program on the master diskette. It is required to boot some 13-sector games.

FORTRAN SYSTEM REQUIREMENTS

The Apple FORTRAN system requires an Apple II or Apple II Plus with 48K of memory, the language system, and at least one disk drive. Use of two drives is recommended for ease of operation and serious program development.

UNFORMATTED I/O IN APPLE FORTRAN

Unformatted I/O is not supported in Apple FORTRAN, as it isn't part of the 1977 ANSI Subset. Statements of the form PRINT*, READ*, or WRITE* are not allowed. All I/O is of the form READ/WRITE(<unit number>,<format identifier>)<iolist>.

MEMORY SPACE FOR FORTRAN

Apple FORTRAN runs under the Pascal operating system, and has approximately 37K available for user programs and data in version 1.0, and over 39K in 1.1, with Swapping toggled on. Please refer to the Pascal Update (addendum).

FORTRAN COPY-PROTECTION

The Apple FORTRAN compiler is protected and cannot be copied. A Bad-Block scan of the FORT2: diskette will show blocks 30, 31, and 32 as bad. DO NOT ATTEMPT TO "FIX" THESE BLOCKS AS THIS WILL RUIN THE FORT2: DISKETTE!! See your local service center if your FORT2: diskette doesn't work.

USING LIBRARY UNITS IN FORTRAN WITH PASCAL 1.1

FORTRAN, running under Pascal version 1.1, does not access the SYSTEM.LIBRARY in order to find the location of each library segment. Therefore, in FORTRAN programs which use library units ($USES <filename>), a stack overflow will occur almost immediately after execution begins.

The International Apple Core is distributing a program which will repair the segment dictionary of each FORTRAN code file, on their April 1981 Disk of the Month (called ATTACH). This Pascal program should be used prior to the first execution of each FORTRAN program, after compilation and linking.

Another method of forcing the operating system to load the locations of each segment is to compile the following Pascal program, and place the code file on your FORTRAN boot disk, and name that code file SYSTEM.STARTUP. The program will be executed automatically during each boot, and will cause the table to be read in.

```
PROGRAM READTABLE;

USES TURTLEGRAPHICS;

BEGIN
END.
```

CREATING FORTRAN DATA FILES

An error has been uncovered in the FORTRAN "CLOSE" statement which causes the operating system to mishandle the disk file space. If a data file is created within a program, closed, then reopened, the system will report that there is no room left on the volume, even if little data was actually written to the file. This is because the OPEN (with STATUS="NEW") statement attempts to reserve all the unused space in the largest available block, and the CLOSE does not release the unused space when the file is closed.

The way to avoid this problem is to "Make" a file on the given disk, of the size you will be needing. For example, your program will be creating a data file (by the name of "Myfile") which will eventually occupy 100 blocks on your data disk. From the Filer, type "M" (for Make), followed by "Myfile[100]". This will create the directory entry, reserving 100 blocks for the data file. The FORTRAN program can then OPEN the data file with STATUS="OLD", and the space will be managed correctly. When the blocks are filled, the expected error messages will occur.

FORTRAN: MAKING THE MOST OF YOUR DISK SPACE

When using FORTRAN in a multiple-drive system, it is not necessary to duplicate system files on both FORT1: and FORT2:. Since the SYSTEM.COMPILER cannot be copied, FORT2: will by necessity have at least one file, but the majority of the space can be free for program development. Follow the procedure for formatting diskettes and transferring files which is described in the Pascal or FORTRAN reference manuals.

Here are two possible FORTRAN configurations which can be used. Your setup will depend on your personal tastes. If you normally do not use, or intend to use, the system work file, you may wish to use this configuration:

FORT1:(boot disk)	FORT2:
SYSTEM.APPLE	SYSTEM.COMPILER
SYSTEM.PASCAL	
SYSTEM.MISCINFO	
SYSTEM.CHARSET	
SYSTEM.EDITOR	
SYSTEM.FILER	
SYSTEM.LINKER	
SYSTEM.LIBRARY	
FORTLIB.CODE	

With this configuration, FORT1: will have only 25 unused blocks, but FORT2: will have 182 blocks available for text and code files! If you exit the Editor by W(riting a named file to FORT2: instead of Updating SYSTEM.WRK, you'll have plenty of room to compile and link your FORTRAN programs.

If you plan to use the system work file, use the following configuration, which will leave the majority of free space on the boot disk. This setup may also be used for writing files, of course. The remaining space on FORT2 can be used for storing files not being used.

FORT1: (boot disk)	FORT2:
SYSTEM.APPLE	SYSTEM.COMPILER
SYSTEM.PASCAL	SYSTEM.LINKER
SYSTEM.MISCINFO	SYSTEM.EDITOR
SYSTEM.LIBRARY	SYSTEM.FILER
	SYSTEM.CHARSET
	FORTLIB.CODE

NOTE: For single-drive systems, please consult your FORTRAN reference manual.

FORTRAN LANGUAGE REFERENCE ERRATA
030-0118-00

Page inside cover
The number A2D0032 is for the diskette, not the manual. The correct number is A2L0032.

Page ix
The page number for the Appendices should be 135 instead of 134.

Page xi
The top paragraph should be deleted because the information is presented on page x.

Page 156
In a multi-drive system it isn't necessary to duplicate the system files on FORT2:. It's better to leave FORT2: empty except for the SYSTEM.COMPILER (which can't be copied anyway) and W)rite your files to FORT2:FILE.TEXT when you leave the Editor and when you compile.

Slot 6 Drive 1	Slot 6 Drive 2
FORT1:	FORT2:
SYSTEM.APPLE	SYSTEM.COMPILER
SYSTEM.PASCAL	SYSTEM.LINKER
SYSTEM.MISCINFO	
SYSTEM.CHARSET	(your files)
SYSTEM.FILER	
SYSTEM.EDITOR	
SYSTEM.LIBRARY	
FORTLIB.CODE	

PASCAL COMPILER OPTIONS

The PASCAL compiler swapping option (*$S+*) should be used if the compile fails and trashes the system due to lack of symbol table space. This option should be the first line of the text, preceding the program statement. Swapping MUST be used in compiling units. Please see page 68 of the Apple Pascal Language Reference Manual.

The PASCAL compiler list option (*$L+*) should not be used with Pascal version 1.0, since this will result in the loss of the code file and possibly of the directory. Instead, direct the list to the printer (*$L Printer:*), the console (*$L Console:*), or to a named file ON ANOTHER VOLUME: for example, (*$L Mydisk:Myfile.text*). Use of the printer is recommended. This problem has been corrected in Pascal 1.1, and will work as described in the Pascal reference manuals.

The (*$U-*) "lex level" compiler option will result in a non-executable codefile, and should not be used for compiling user programs. This option is intended for working on the system level, and requires files and information not available to the user.

Compiler options in Pascal version 1.0 are required to be in capitals. In version 1.1, capitals and lower case are interchangeable.

UPPER AND LOWER CASE IN PASCAL 1.1 AND PILOT

Apple Pascal version 1.1 supports upper and lower case through the standard Apple keyboard and display. When enabled, upper case will be displayed as inverse video, with lower case as normal video. In this mode, all input will be lower case (normal video) unless a CTRL-W or CTRL-E is typed.

Version 1.1 also has the capability of sensing the shift key if the key is wired to SW2 on the game I/O connector. In this case, pressing the shift key will close SW2, and Pascal will shift to upper case as long as the shift is held down.

A number of game I/O devices can interfere with this function, however, preventing normal upper/lower case from being used, despite the control functions typed. The SW2 sensing will override the control characters used from the keyboard. If Pascal 1.1 or Apple Pilot will not generate anything except upper case, try removing whatever device has been plugged into the game I/O port, and try inputting again.

PASSING STRINGS TO AND FROM FUNCTIONS

Strings can be passed to and from functions or assembly language routines by using formal parameters (pass by address). An example of passing a string to a function is listed below. Since functions cannot return a non-scalar variable, a procedure with formal parameters works just as well.

```
PROGRAM PASS;

VAR S: STRING;

PROCEDURE STUFFIT (VAR S: STRING);     (* the "VAR" is important *)
 BEGIN
  S:='THIS IS A STRING';
 END;

BEGIN
 S:='';    (* just to prove the string is empty *)
 WRITELN (S);
 STUFFIT (S);
 WRITELN (S)
END.
```

PASCAL PERIPHERAL SLOT ASSIGNMENTS

The Apple Pascal system has specific slot assignments for various peripheral devices. Slot assignments are listed on page 277 of the Pascal Operating System manual.

The Apple Pascal system comes configured to work with Apple interface cards. To use other types of interface cards, it may be necessary to modify the system or add additional driver routines. Pascal version 1.1 has provision for attaching additional peripheral drivers through the use of the ATTACH utility. This utility is being supplied by the International Apple Core to all member users groups. All such modifications are at the user's own risk.

If a com card, serial card, or DC Hayes Micromodem card is placed in slot #3, the Pascal system will not appear to boot correctly on the standard Apple video. This is due to the fact that Pascal has recognized a card in slot #3, which is reserved for an external terminal, and is attempting to communicate with the terminal instead of the default CONSOLE and SYSTERM (video and keyboard). Other cards may also cause this effect, which can be easily remedied by removing the card from slot #3.

NESTING PASCAL PROCEDURES AND FUNCTIONS

Apple Pascal will allow nesting of procedures inside functions and functions inside procedures. Procedures and functions may be nested up to 15 deep.

Segment procedures and segment functions may be nested up to six levels deep, although the resultant code will not be nested. Please refer to the Apple Pascal Language Reference Manual for more details on segmentation.

PASCAL MEMORY SPACE

Approximately 39K bytes are available for program execution in the Apple PASCAL system, version 1.0.

The maximum space in Pascal version 1.1 is about 37K with swapping off, and nearly 40K with swapping on. Please refer to the Apple Pascal Update addendum for further information about the command level swapping option.

The Pascal MEMAVAIL function does not always return the correct value in version 1.0. In this instance, the correct value for MEMAVAIL can be returned if the function is placed inside a procedure, instead of being called directly from the main section of the program. MEMAVAIL works properly in Pascal 1.1.

PASCAL IMMEDIATE MODE

Apple Pascal does not have an "immediate mode" where program instructions can be executed individually, since it is a compiled language. The command line is used for accessing specific system functions such as the Compiler, Editor, Filer, Assembler, or Linker.

PASCAL TURNKEY SYSTEM

A Pascal turn-key system can be created by transferring the desired program codefile to the BOOT diskette, and changing the file name to SYSTEM.STARTUP. If no file by that name is present on the boot volume, the system will default to the command line.

PASCAL EXTERNAL REFERENCES

The use of "External" linking in Apple Pascal is restricted to assembly routines. Pascal procedures which are to be compiled separately should be put into a library unit and accessed through the "USES" command.

PASCAL INVERSE VIDEO

Apple Pascal version 1.1 supports normal and inverse video on the standard 40-column Apple screen, which are used to indicate lower and upper case, respectively. Flashing mode is not supported. Program control of normal and inverse characters is not supported.

Additional information about upper/lower case support is listed on pages 9-10 of the Addendum to the Apple Pascal Operating System Reference Manual.

MAKING A COPY OF "BASICS"

The language system and DOS 3.3 diskette named BASICS is a 16 sector PASCAL format diskette and should be copied using the PASCAL Filer Transfer or the DOS 3.3 COPY program. FID will not be able to copy this diskette.

TTLOUT

The Pascal TTLOUT procedure in the version 1.0 Applestuff unit in the SYSTEM.LIBRARY does not function properly. The routine can be assembled and linked externally, using the listing and example shown on pages 100-106 of the Apple Pascal Reference Manual, version 1.0.

This problem has been corrected for version 1.1, so that TTLOUT will function correctly from Applestuff. If desired, TTLOUT can still be assembled and linked externally, to avoid using all of Applestuff.

PASCAL SCAN FUNCTION

The Pascal SCAN function scans a range of memory, searching for a specified target. The function takes the form

```
SCAN (LIMIT, PEXPR, SOURCE)
```

where LIMIT is the number of bytes to be scanned, PEXPR defines the target of the scan, and SOURCE is the variable being scanned. Pages 51-52 of the Pascal Language Manual describe this function in greater detail.

SCAN can be used on any variable except file types. When scanning a string (S, for example), SOURCE should be specified as S[1], to avoid having SCAN also look at the length parameter stored in S[0], which will usually yield an erroneous value.

REBOOTING PROBLEMS IN PASCAL

If recurrent rebooting is a problem in your Pascal system, especially when typing in the editor, it may be due to striking more than one key at the same time. Try slowing down to allow the previous key to be released before typing the next one. When typing quickly, many people do not completely release one key before striking the next. If you attempt to hold down more than one key at a time, the encoder chip sees the logical "AND" of the keys currently being held down. The AND operation results in bits being turned off, so it is quite likely that many combinations of keys will result in an ASCII 0 (NUL), which is a BREAK in Pascal.

The reason this doesn't happen in Basic is that an ASCII 0 from the keyboard has no function in the Basic system. If slowing down doesn't help the problem, it may be due to memory errors, and it would be wise to have the hardware checked out.

CHAINING IN PASCAL

Pascal version 1.0 does not allow one program to call (run) another program. A program containing segment procedures can be used to simulate chaining.

Version 1.1 of Apple Pascal contains the library unit CHAINSTUFF, which will allow chaining without the use of segment procedures. Please note that this type of chaining does not preserve the variable space. Refer to the Apple Pascal Language Manual addendum for more information about chaining.

PASCAL BOOLEANS

The boolean operator, NOT, does not invert the boolean variable correctly in Pascal version 1.0. For example, NOT FALSE doesn't equal TRUE. It is suggested that the 'NOT' be replaced by

```
IF A=TRUE THEN A:=FALSE ELSE A:=TRUE
```

when working with booleans in version 1.0.

Booleans will work correctly in version 1.1. Please refer to pages 8-9 of the Pascal Update addendum for a description of the corrections made to booleans.

PASCAL TRUNC AND ROUND FUNCTIONS

The Pascal TRUNC and ROUND functions are present in Apple Pascal. The ROUND function is of the form I:=ROUND(R) where I is an integer and R is a real number. Although the compiler will accept other forms of this command, this is the only legitimate form.

TRUNC takes the form I:=TRUNC(R) where I is an integer and R is a real number, and I:=TRUNC(L) where L is a long integer. In all cases, integer values must be in the range -32767 to +32767. The compiler will accept other forms of this function, but these are the only correct forms according to UCSD definitions.

PROPOSED ISO STANDARDS FOR PASCAL

For information on the proposed ISO standard for Pascal, please write to:

Dr. Carol Sledge, Chairman X3J9 Committee
On-Line Systems Inc.
115 Evergreen Heights Drive
Pittsburgh, PA 15229
(412) 931-7600

The message "ERROR: WRITING OUT THE FILE. PRESS <SPACEBAR> TO CONTINUE." can be generated in one of three ways. First, file names are restricted to 10 characters, not including the volume name prefix, or the .TEXT suffix. A name longer than that will generate the error immediately, without accessing the disk drive.

Secondly, and more commonly, if there is insufficient room on the disk to save the new file, this message will be reported. When updating the workfile, Pascal always saves the new copy of the file before removing the previous copy, so the possibility of insufficient space may occur at this time. The last, and by far the least common, is that if the directory is full (77 entries), even if there is enough room on the disk, no additional files can be saved.

A "No room on volume" error, or "code write error" can be generated in Pascal if there isn't sufficient room to write the file to the specified volume. This error can occur in the Filer, Compiler, Assembler, or Linker, and can usually be corrected by using the Filer "Krunch" option (to consolidate unused blocks), or removing unnecessary files.

"Code write error" can occur in the Assembler (and occasionally in the Compiler) when creating a code file on the boot disk. The Assembler creates a temporary file called LINKER.INFO on the boot disk, that stores linkage information which is later written into the final block of the code file. During an assembly when the code file is directed to the boot disk, the largest unused block of disk space is opened for the code file. If the disk has been Krunched recently, this may result in no space available for opening LINKER.INFO, and thus the error message. In this case, it would be better NOT to Krunch the disk, but if that's already been done, simply "Make" an 8-block file on the boot disk, then Make a 1-block file (names unimportant). Then Remove the 8-block file. This separates the space into two distinct areas so that no conflict arises. This can also hold true for the Compiler, but much less often, since the only time LINKER.INFO is created is during very large compiles when the system needs to swap some information out temporarily.

If an error occurs when using the include file option, "Make" a 4-block file on the volume where the compiler resides, and name it SYSTEM.SWAPDISK. This allows the compiler to swap out some information in order to read in the new directory in preparation for the include file.

The Pascal (version 1.0) compiler error #407 ("Too Many Libraries") can usually be avoided by changing the length of the text file being compiled. The exact cause of the error is not known. Pascal version 1.1 has corrected this problem.

Run-time errors generate a message plus a set of numbers that indicate which instruction was being executed when the error occurred. "S" stands for "SEGMENT", "P" for "PROCEDURE", and "I" for "INSTRUCTION COUNT". These can be correlated with the textfile by using the System List option (*$L+*) or (*$L<filename>*) when doing a compile. This will produce an annotated listing of the text, with the S, P, and I numbers included.

For example, here is a very simple program which has been compiled with the listing option:

```
 (1)      (2)     (3:4)       (5)<--TEXT---------->
  1        1       1:D         1 (*$LPRINTER:*)
  2        1       1:D         1 PROGRAM EXAMPLE;
  3        1       1:D         3
  4        1       1:D         3 VAR S:STRING;
  5        1       1:D        44
  6        1       1:0         0 BEGIN
  7        1       1:1         0   READLN(S);
  8        1       1:1        21   WRITELN(S)
  9        1       1:0        40 END.
```

KEY:

(1) Line number of text
(2) Segment number (S#) - S# values which do not appear in your listing indicate that the error has occurred in that segment of the operating system. S#0 is SYSTEM.PASCAL, and 17-31 are usually SYSTEM.LIBRARY segments.
(3) Procedure number (P#)- the procedure number within the segment designated by the S#.
(4) Nesting level (D=declaration)
(5) Instruction count (I#) - this is the instruction count from the beginning of the procedure. The number indicates the count at the beginning of each line, so a value between two lines simply means that the error occurred in the middle of the line.

A word of warning: do NOT use L+ as the listing option in Pascal version 1.0, as this will result in the loss of your code file and possibly your operating system. Instead, name a disk file that will be placed on a DIFFERENT volume from the destination of the code file, or better yet, use (*$LPRINTER:*) to put the listing directly onto your printer.

PASCAL REAL NUMBER FORMAT

Real numbers, in UCSD Pascal, are floating point numbers in the range of +/- 1.17550E-38 to +/-3.40282E+38. Real numbers use four bytes (2 words). The binary representation is similar to the proposed IEEE standard for floating point numbers:

```
 31   30 . . . . . 23  22 . . . . . . . . . . 0  <== 32 bits
SIGN      EXPONENT               MANTISSA
```

"Mantissa" is the name given to the decimal portion of a number which is expressed in scientific (exponential) notation. The "exponent" indicates the power to which the mantissa is raised. The exponent is represented in base 2 (2^N). In decimal, the number 3 x 10^2 can be seen as a mantissa of 3, an exponent of 2, in base 10 (decimal).

The sign bit refers to the sign of the mantissa, and is 0 if positive, 1 if negative. The exponent is "offset" by 127; that is, a value of 127 in the exponent field corresponds to an exponent of 0. Similarly, if the value is 1, the exponent is -126, and if the field is 254, the exponent is +127. A value of 0 indicates that the real number is 0.

The mantissa of the real number is stored in normalized format in bits 0-22. "Normalizing" a number means adjusting it so that the highest bit is significant (a 1). The exponent indicates how many times (and in which direction) the value was shifted during normalization.

Notice that the MSB of the mantissa of any non-zero number which has been normalized is always a one. The number zero can be treated as a special case by simply setting the exponent to zero. So, to gain additional precision, the mantissa has an implied "1" which is not stored, resulting in a functional 24-bit mantissa, even though only 23 bits are actually used. This gives slightly more than 6 decimal places (single precision) accuracy.

Real numbers may be formatted for output using field-width designations. As described on pages 36-37 of the Apple Pascal language Reference Manual, the output specification has the following form:

```
REAL : FIELDWIDTH : FRACTIONLENGTH
```

where FIELDWIDTH is the minimum number of characters to be written, including the decimal point (default=1), and FRACTIONLENGTH is the number of digits to be written after the decimal place (default=5). Thus, a field specification of R:8:3 would indicate the real variable R is to be printed within a field size of 8 total, with 3 of those digits appearing to the right of the decimal. A FRACTIONLENGTH of zero is not allowed.

If the field size needed to display the variable accurately exceeds the formatting specification, the formatting will be ignored. If the size is smaller than FIELDWIDTH, the field will be padded with blanks to the left of the variable (right-justification).

USING RESIDENT AND NOLOAD COMPILER OPTIONS WITH INTRINSIC UNITS

Intrinsic units can be treated as segments (overlays) in Apple Pascal version 1.1 by using the noload (*$N+*) and the resident (*$R <unit name>*) compiler options. Units specified by these options are read in from the disk only when a reference is made to code contained within the unit. When the calling procedure is exited, and all active references to the unit have been resolved, the unit is "swapped out", and the memory range can be used for other code segments. For more details on the use of these options, refer to pages 66-67 of the Pascal Language Reference Manual, and the Language Manual addendum.

When using the noload option, the unit code will remain in memory until the calling procedure is exited. This may result in a stack overflow (out of memory) if the unit is called repeatedly from inside the same procedure, without exiting. This is a known problem, and can be avoided by specifying the unit as "Resident" within that procedure or segment. Calling the unit from within a different procedure will also remedy this. The following examples illustrate this situation:

EXAMPLE A: WILL GENERATE AN OUT OF MEMORY STACK OVERFLOW ERROR BY REPEATEDLY LOADING APPLESTUFF.

```
PROGRAM EXAMPLEA;
USES APPLESTUFF;
VAR I: INTEGER;
BEGIN   (* MAIN PROGRAM *)
  (*$N+*)
  WRITELN ('NUMBER OF WORDS AVAILABLE AT START OF PROGRAM : ',MEMAVAIL);
  FOR I:=1 TO 100 DO
    BEGIN
      WRITELN (I,' ',MEMAVAIL,' WORDS');
      NOTE (25,1) (* APPLESTUFF MUSIC ROUTINE *)
    END;
  WRITELN ('PROGRAM COMPLETED SUCCESSFULLY')
END.   (* MAIN PROGRAM *)
```

EXAMPLE B: USES RESIDENT OPTION TO PREVENT APPLESTUFF FROM BEING RELOADED.

```
PROGRAM EXAMPLEB;
USES APPLESTUFF;
VAR I: INTEGER;
BEGIN  (* MAIN PROGRAM *)
  (*$N+*)
  (*$R APPLESTUFF *)
  WRITELN ('NUMBER OF WORDS AVAILABLE AT START OF PROGRAM : ',MEMAVAIL);
  FOR I:=1 TO 100 DO
    BEGIN
      WRITELN (I,' ',MEMAVAIL,' WORDS');
      NOTE (25,1) (* APPLESTUFF MUSIC ROUTINE *)
    END;
  WRITELN ('PROGRAM COMPLETED SUCCESSFULLY')
END.   (* MAIN PROGRAM *)
```

EXAMPLE C: USES INDIVIDUAL PROCEDURE TO CALL APPLESTUFF ROUTINE, WHICH ALLOWS UNIT TO BE RELEASED AND THEN RELOADED.

```
PROGRAM EXAMPLEB;
USES APPLESTUFF;
VAR I: INTEGER;
PROCEDURE PLAY;
  BEGIN
    WRITELN (I,' ',MEMAVAIL,' WORDS');
    NOTE (25,1) (* APPLESTUFF MUSIC ROUTINE *)
  END;
BEGIN  (* MAIN PROGRAM *)
  (*$N+*)
  WRITELN ('NUMBER OF WORDS AVAILABLE AT START OF PROGRAM : ',MEMAVAIL);
  FOR I:=1 TO 100 DO PLAY;
  WRITELN ('PROGRAM COMPLETED SUCCESSFULLY')
END.   (* MAIN PROGRAM *)
```

THE KEYPRESS FUNCTION

KEYPRESS is a widely used function which resides in the unit APPLESTUFF. In many cases, it is the only routine being called from the unit, so it would be more efficient to refer to the routine without also "using" the rest of the unit.

The attached listing is the function "KEYPRESS", which can be assembled for linkage to your host program as an "external" function. Follow the instructions provided with the example shown on pages 136-182 of the Pascal Operating System Reference Manual.

```
        .FUNC KEYPRESS,0    ;0 words of parameters passed
;************************************************
;*
;*   FUNCTION KEYPRESS: BOOLEAN; EXTERNAL
;*
;************************************************
;
RETURN  .EQU 0              ;Storage for return address
CONCKVEC.EQU 0BF0A          ;Fixed address in BIOS
RPTR    .EQU 0BF18          ;Fixed buffer pointer
WPTR    .EQU 0BF19          ;Fixed buffer pointer
VERSION .EQU 0BF21          ;System version number
KEYBOARD.EQU 0C000          ;Keyboard hardware
CONCK   .EQU 0FF5C          ;Way to get CONCK in old system

        PLA
        STA RETURN
        PLA
        STA RETURN+1
        PLA
        PLA
        PLA
        PLA                 ;Pop 4 bytes stack bias for function
        LDA #0
        PHA                 ;Return MSB zero
        LDA KEYBOARD
        BMI TRUE
        LDA VERSION
        BNE $1              ;Jump if not original Pascal version
        JSR CONCK
        JMP $2
$1      JSR CONCKVEC        ;Check console
$2      LDA RPTR
        CMP WPTR            ;Char in buffer?
        BEQ EMPTY
TRUE    LDA #1              ;Yes, return KEYPRESS=TRUE
        BNE KPDONE          ;Always taken
EMPTY   LDA #0              ;No, return KEYPRESS=FALSE
KPDONE  PHA                 ;Push LSB result
        LDA RETURN+1
        PHA                 ;Restore return address
        LDA RETURN
        PHA
        RTS
        .END
```

THE DEMO

This brief program illustrates the use of KEYPRESS as an externally linked routine. Follow the instructions given on pages 100-106 of the Pascal reference manual for assembling and linking external code.

```
PROGRAM PRESSTEST;

VAR I: INTEGER;

FUNCTION KEYPRESS: BOOLEAN; EXTERNAL;

BEGIN
I:=0;
REPEAT
  WRITELN (I);
  I:=I+1;
UNTIL KEYPRESS
END.
```

PASCAL UNITS

Modular programming means the separation of procedures and functions, or groups of them, from the main program. Source language modules are called units, and are incorporated in libraries for use with Pascal programs. Units may consist of procedures, functions, or a combination of these, in Pascal and assembly language.

Separate compilation has several advantages in the development of any program, because it allows you to approach the task as a group of smaller tasks which are linked together in a logical manner. The host program must contain a USES statement to utilize routines from the unit.

There are two principal kinds of units: Regular units, and Intrinsic units. Separate units are not included in the Apple Pascal implementation, and will not be discussed.

REGULAR UNITS

When a host program USES a regular unit, the unit's code is physically inserted into the host's codefile by the Linker. Once linked, the files need not be relinked unless either the unit or the host program is modified and recompiled.

Regular units, since they become part of the host file, may have references to file names. A regular unit can use another regular unit, or it may use an intrinsic unit.

Regular units may be installed in the SYSTEM.LIBRARY, or in any other library file. If installed in an alternate library, the uses statement should include the compiler option $U <library name> before the unit name.

RESTRICTION: In version 1.0 of the Apple Pascal system, regular units may NOT use intrinsic units.

INTRINSIC UNITS

An intrinsic unit is pre-linked; that is, it contains sufficient information to allow the host program to use it without invoking the Linker. The code for an intrinsic unit remains in the SYSTEM.LIBRARY, and is loaded into memory from the library when the host program is executed. This keeps the size of the host program down; it also allows the unit and host program to be modified and recompiled individually without the need to relink.

Intrinsic units must be installed in the SYSTEM.LIBRARY. An intrinsic unit can USE another intrinsic unit, and may contain references to file names.

RESTRICTION: Intrinsic units may not use regular units or have references to file names in Pascal version 1.0.

ASSEMBLY ROUTINES AS PART OF UNITS

Assembly language routines may be placed into library units. With intrinsic units, the unit is compiled, the machine language routines assembled, then the assembled code is linked to the unit PRIOR to installing the unit into the SYSTEM.LIBRARY.

Regular units may also contain machine language routines; however, these routines are NOT linked to the unit before it is installed in the library. Instead, the host program, the unit, and the assembly routines are linked together at the same time.

AN ADDITIONAL NOTE ON THE CONSTRUCTION OF A UNIT

Any unit which does not contain at least one procedure in the INTERFACE section is not allowed to have an IMPLEMENTATION section. You must include the initialization BEGIN..END, however.

Procedures listed in the INTERFACE section are public to the host program as well as to the unit. Those procedures listed only in the IMPLEMENTATION section cannot be accessed by the calling program (private). If no procedures are listed publicly, then none can be called from the host, and the IMPLEMENTATION section is not allowed.

The following example is designed to illustrate the general structure of a unit. The line numbers at the left of the page are for reference, and are not part of the actual structure. Discussion of the format will refer to the line numbers.

```
 1:  (*$S+*)
 2:  UNIT < name >; INTRINSIC CODE xx (DATA yy);

 3:  INTERFACE

 4:    USES < name of unit to be used >;
       CONST < definitions >;
       TYPE < definitions >;

 5:    VAR < definitions >;

 6:    PROCEDURE ONE (I:Integer);
       PROCEDURE TWO (I:Integer);  (* An external procedure *)
       FUNCTION THREE (I:Integer) : Integer;
       FUNCTION FOUR (I:Integer) : Integer;  (* An external function *)

 7:  IMPLEMENTATION

 8:    CONST < definitions >;
       TYPE < definitions >;

 9:    VAR < definitions >;

10:    PROCEDURE ONE;
         BEGIN
          .
         END;

       PROCEDURE TWO; EXTERNAL;

       FUNCTION THREE;
         BEGIN
          .
         END;

       FUNCTION FOUR; EXTERNAL;

11:    BEGIN
       (* initialization section *)
       END.
```

DISCUSSION

1: The swapping option is required when compiling ANY unit, regardless of its size. This is the most common cause of compiler failures when working with units. The option should be the first line of text, preceding the UNIT header and any other compiler option.

2: The word INTRINSIC is required if the unit is to be intrinsic. DATA is optional, used only if a data segment is necessary (see #5). Regular units use only the UNIT < name > portion of this line.

3: INTERFACE is required. It defines the start of this section, which must contain some entries (a totally empty INTERFACE section is not allowed). This section contains the information which is public to the host program as well as the unit (and visible from the LIBMAP program).

4: These entries are optional. If used, they are public (see #3). If a nested unit is used, it MUST be declared at this point, and the host program must also declare it in its USES statement (see pages 75-81 in the Pascal Language Reference Manual).

5: VARs are also optional. If VARs are used in an intrinsic unit, a DATA segment MUST be declared, since these are global variables.

6: These are the public statements of the unit's procedures and functions. All parameters MUST be declared in this section, and must not appear in the IMPLEMENTATION section. EXTERNAL may not be specified at this point (see #10).

7: IMPLEMENTATION is required. This section contains the code generating statements, and any private types or variables to be declared. If an item is declared in this section only, it is local to the unit and cannot be accessed from the host program directly, and is not visible to LIBMAP.

8: CONST and TYPE definitions are optional. If included at this point, they are private.

9: VARs are optional, and are private if defined at this point. A DATA segment is required if global VARs are specified in the IMPLEMENTATION section of an intrinsic unit, even though they are private.

10: These are the actual code generating procedures. Procedures and functions already declared in the INTERFACE section may not have parameters listed here, as this would be a duplication. EXTERNAL references are specified at this time.

11: This is the initialization section. Code in this portion is optional, but the BEGIN and END statements must be present in any event. The final END statement must be followed by a "." to indicate the end of the text. Code placed in the initialization section will be executed immediately upon access to the unit (through the USES statement in the host program), and ignored thereafter. An example of this is TURTLEGRAPHICS, where the initialization code is responsible for allocating the hi-res page so that variables will not be lost.

PASCAL LONG INTEGER COMPARES - VERSION 1.0 ONLY

An error in the implementation of Pascal long integers results in a stack crash during a compare operation in version 1.0. The attached program will repair the LONGINTEGER module in the 1.0 SYSTEM.LIBRARY. The problem has been corrected in Pascal version 1.1, so that long integers will work as expected.

NOTE: THIS PROGRAM IS PROVIDED FOR VERSION 1.0 SYSTEMS ONLY, AND SHOULD NOT BE USED ON VERSION 1.1.

```
(*$I-*)
PROGRAM FIXCOMPARE;

TYPE  DISKINFO = RECORD
                   DADDR: INTEGER;
                   LENG : INTEGER
                 END;

      NAME = PACKED ARRAY [1..8] OF CHAR;

      SEGDIC = RECORD
                 DINFO: ARRAY [0..15] OF DISKINFO;
                 SEGNAME: ARRAY [0..15] OF NAME;
                 FILLER: ARRAY [1..416] OF INTEGER
               END;

      BLOCK = PACKED ARRAY [0..511] OF 0..255;

VAR  I,J: INTEGER;
     NOTFOUND: BOOLEAN;
     F: FILE;
     S: STRING;
     BLOCKZERO: SEGDIC;
     DATA: BLOCK;

PROCEDURE READERROR;
BEGIN
  WRITE ('BAD BLOCK IN LIBRARY');
  EXIT (PROGRAM)
END;

BEGIN
  NOTFOUND:=TRUE;
  REPEAT
    WRITE ('NAME OF LIBRARY FILE:');
    READLN (S);
    RESET (F,S);
  UNTIL EOF OR (IORESULT=0);
  IF BLOCKREAD (F,BLOCKZERO,1,0)<>1 THEN READERROR;
  FOR I:=0 TO 15 DO
    BEGIN
      IF BLOCKZERO.SEGNAME[I]='LONGINTI' THEN
        BEGIN
```

```
          NOTFOUND:=FALSE;
          J:=BLOCKZERO.DINFO[I].DADDR+1;
          IF BLOCKREAD (F,DATA,1,J)<>1 THEN READERROR;
          IF DATA[495]=244 THEN
            IF DATA[494]=208 THEN
              BEGIN
                WRITELN ('LONG INTEGER PATCH BEING MADE');
                DATA[494]:=240;
                IF BLOCKWRITE (F,DATA,1,J)=1 THEN
                  WRITELN ('PATCH COMPLETE')
                ELSE
                  WRITELN ('ERROR WHILE WRITING PATCH, SEGMENT ',I);
              END
            ELSE
              IF DATA[494]=240 THEN
                WRITELN ('SEGMENT ',I,
                         ' - LONG INTEGER UNIT HAS ALREADY BEEN FIXED')
              ELSE
                WRITELN ('CAN''T RECOGNIZE LONGINT UNIT IN SEGMENT ',I)
          ELSE
            WRITELN ('CAN''T RECOGNIZE LONG INTEGER UNIT IN SEGMENT ',I)
        END
    END;
  IF NOTFOUND THEN WRITELN ('NO LONG INTEGER UNIT FOUND')
  ELSE
    WRITELN ('PROGRAM COMPLETE')
END.
```

CONVERTING STRINGS TO NUMERIC VARIABLES

Apple Pascal doesn't allow input editing for integer or real variables. This can cause a lot of trouble if the wrong data is entered by mistake: either the program is stuck with bad data, or worse yet, the system may crash.

STRINGSTUF is an intrinsic unit which is designed to avoid this problem. All data is entered in the form of strings, then converted to the appropriate data format by the unit. This allows editing while the data is in string form.

Note: The STRINGSTUF unit and accompanying demo program are designed to run in the new release of Pascal, version 1.1 only. Long integers are not supported in STRINGSTUF.

THE UNIT

STRINGSTUF may be located in any unused segment number 17-31. Install in SYSTEM.LIBRARY following the instructions in the Pascal Operating System Reference Manual, pages 186-193.

```
 (*$S+*)
(*$LPRINTER:*) (* Get a compile listing..optional *)

 UNIT STRINGSTUF;  INTRINSIC CODE 26;

 INTERFACE


   TYPE STRING255=STRING[255];

   FUNCTION STRFP (VAR STR:STRING255; VAR FP:REAL): BOOLEAN;
   FUNCTION STRINT (VAR STR:STRING255; VAR INT:INTEGER): BOOLEAN;


 IMPLEMENTATION


 FUNCTION STRFP;  (* String to Real *)

 CONST MAXREAL=1.70E37;  (* Max/10 *)
      MINREAL=1.2E-37;  (* Min/10 *)

 VAR DEC,DEX,EDP,INX,LEN: INTEGER;
    DP,EX,IM,MN,MX,SN: BOOLEAN;
    CH: CHAR;
    NUMERIC,EXPONENT,MODIFIER: SET OF CHAR;

   PROCEDURE TERMINATE;
   VAR I: INTEGER;
   BEGIN
      IF MX THEN DEX:=-DEX;
      EDP:=EDP+DEX-DEC;
      IF EDP<0
         THEN FOR I:=1 TO -EDP DO
            IF FP>=MINREAL THEN FP:=FP/10.0
               ELSE FP:=0  (* Underflow => 0 *)
         ELSE FOR I:=1 TO EDP DO
            IF FP<=MAXREAL THEN FP:=FP*10.0
               ELSE EXIT (STRFP);  (* Overflow *)
      IF MN THEN FP:=-FP;
      STRFP:=TRUE;
      EXIT (STRFP)  (* Successful conversion *)
   END;
```

```
  PROCEDURE SEARCH;
  BEGIN
     WHILE INX<=LEN DO
        IF STR[INX] IN NUMERIC
           THEN BEGIN
  (*$R-*)
              WHILE (INX>1) AND (STR[INX-1] IN EXPONENT+MODIFIER)
  (*$R+*)
                 DO INX:=INX-1;
              EXIT(SEARCH)  (* Found start of number *)
           END
           ELSE INX:=INX+1;
     EXIT (STRFP)  (* Non-numeric string *)
  END;

BEGIN (*STRFP*)
  NUMERIC:=['0'..'9'];
  EXPONENT:=['E','e'];
  MODIFIER:=['+','-','.',',','];
  DP:=FALSE; EX:=FALSE; IM:=TRUE;
  MN:=FALSE; MX:=FALSE; SN:=FALSE;
  DEC:=0; DEX:=0; EDP:=0; INX:=1;
  LEN:=LENGTH(STR); FP:=0;
  STRFP:=FALSE;
  SEARCH;  (* Find start of number *)
  WHILE INX<=LEN DO BEGIN
     CH:=STR[INX];
     IF CH IN NUMERIC+EXPONENT+MODIFIER
        THEN BEGIN
           IF CH IN NUMERIC
              THEN IF EX
                 THEN BEGIN
                    IF DEX<1000 THEN
                       DEX:=DEX*10+ORD(CH)-ORD('0');  (* Exponent *)
                    SN:=TRUE
                 END
                 ELSE BEGIN
                    IF FP<1.0E8
                       THEN FP:=FP*10+ORD(CH)-ORD('0')  (* Mantissa *)
                       ELSE EDP:=EDP+1;
                    IF DP THEN DEC:=DEC+1;  (* Digits to right of DP *)
                    IM:=FALSE;
                    SN:=TRUE
                 END
              ELSE CASE CH OF
                 '+': IF SN THEN TERMINATE  (* Duplicate '+' sign *)
                         ELSE SN:=TRUE;
                 '-': IF SN THEN TERMINATE  (* Duplicate '-' sign *)
                         ELSE BEGIN
                            IF EX THEN MX:=TRUE
                               ELSE MN:=TRUE;
                            SN:=TRUE
                         END;
                 '.': IF DP OR EX THEN TERMINATE  (* Duplicate '.' *)
```

```
                        ELSE DP:=TRUE;
                  'E','e': IF EX THEN TERMINATE  (* Duplicate 'E' *)
                        ELSE BEGIN
                           IF IM THEN FP:=1.0;  (* Implied mantissa *)
                           EX:=TRUE;
                           SN:=FALSE
                        END;
               END; (*CASE*)
            INX:=INX+1
         END
         ELSE TERMINATE  (* End of number *)
      END;
  TERMINATE  (* End of string *)
END;


FUNCTION STRINT;  (* String to Integer *)

VAR FP: REAL;

BEGIN
  STRINT:=STRFP (STR,FP);  (* First convert to real *)
  IF ABS(FP)<=MAXINT
     THEN INT:=ROUND(FP)  (* then round to integer *)
     ELSE BEGIN
        STRINT:=FALSE;  (* Integer out of range *)
        INT:=0
     END
END;


BEGIN  (* Unit Initialization *)
END.
```

THE DEMO

This program illustrates the use of STRINGSTUF. The compiler $V- option is required to override the normal string length checking.

```
PROGRAM STRINGTEST;

USES STRINGSTUF;  (* tests StringStuf library unit *)

VAR INPUT,STR: STRING;
   INT: INTEGER;
   FP: REAL;

BEGIN
  PAGE (OUTPUT);
  WRITELN ('STRINGSTUF STRING => NUMERIC CONVERSION:');
  REPEAT
     WRITELN;
     WRITE ('STRING : ');
     READLN (INPUT);
(*$V-*)
     IF STRFP (INPUT,FP) THEN
       BEGIN
         WRITELN ('   REAL: ',FP);
         IF STRINT (INPUT,INT)
           THEN WRITELN ('INTEGER:  ',INT)
             ELSE WRITELN('INTEGER:  OUT OF RANGE.')
       END
         ELSE WRITELN('NO NUMERIC VALUE IN STRING.');
(*$V+*)
  UNTIL INPUT=''
END.
```

PASCAL UNIT FOR SILENTYPE PROCEDURES

Appendix D (pages 53-59) of the Silentype reference manual indicates that all the parameter procedures can be placed into an intrinsic library unit. This is a handy and valuable unit to have if you plan to use the Silentype in Pascal.

To extend the usability of the Silentype unit, two procedures have been added to this listing, and another has been changed. These procedures will be discussed here; please refer to the Silentype manual for a description of the remainder of the routines.

PROCEDURE SETPAGE (PAGE: INTEGER)

This procedure will allow you to change from one hi-resolution graphics screen to the other, although only the first hi-res page is used in the Pascal system.
Values: 1=first page
2=second page

PROCEDURE COLDSTART

The COLDSTART procedure will reset all the Silentype parameters to the default values as specified in the printer manual. This procedure is equivalent to turning off the power to the printer.
Values: none required.

PROCEDURE WARMSTART

The RESTORE procedure as listed in the Silentype manual has been revised and renamed WARMSTART. Included in the revision is the SETPAGE procedure. This procedure differs from COLDSTART in that the parameters set by WARMSTART are user-definable; that is, they may reflect the particular setup that you find most applicable to your needs. (For instance, you may want a darkness setting of 3 instead of 5.)

INITIALIZATION CODE

The initialization code consists of a call to the COLDSTART procedure. This will be executed when the unit is first brought into the system from the library. This will insure that the printer does not have any unwanted values remaining from previous usages in either Pascal or in Basic. If desired, you may change this to WARMSTART or other code, or leave the initialization section empty (but the BEGIN and END must be present).

INSTALLING THE UNIT

The Silentype unit is ready to type in, compile, and install in your copy of SYSTEM.LIBRARY. You may wish to change the INTRINSIC CODE value, or the contents of WARMSTART to suit your individual needs. Refer to pages 186-193 of the Apple Pascal Operating System Reference manual for instructions on the LIBRARY program which will place the compiled Silentype unit into your SYSTEM.LIBRARY.

```
(*$S+*)
 UNIT SILENTYPE; INTRINSIC CODE 27;

 INTERFACE

  PROCEDURE ROMENABLE;
  PROCEDURE SETBYTEVALUE (LOC,VALUE: INTEGER);
  FUNCTION BYTEVALUE (LOC: INTEGER): INTEGER;
  PROCEDURE SEND (CH: CHAR);
  PROCEDURE SETUNIDIRECT;
  PROCEDURE SETBIDIRECT;
  FUNCTION UNIDIRECT: BOOLEAN;
  PROCEDURE SETNEGATIVE;
  PROCEDURE SETPOSITIVE;
  FUNCTION NEGATIVE: BOOLEAN;
  PROCEDURE SETPAGE (PAGE: INTEGER);
  PROCEDURE SETDARK (DARKNESS: INTEGER);
  FUNCTION DARK: INTEGER;
  PROCEDURE SETFORM (LENGTH: INTEGER);
  FUNCTION FORM: INTEGER;
  PROCEDURE SETSPACE (LENGTH: INTEGER);
  FUNCTION SPACE: INTEGER;
  FUNCTION LEFTMARGIN: INTEGER;
  FUNCTION RIGHTMARGIN: INTEGER;
  PROCEDURE SETLEFTMARGIN (POSITION: INTEGER);
  PROCEDURE SETRIGHTMARGIN (POSITION: INTEGER);
  PROCEDURE PRINTBUFFER;
  PROCEDURE CLEARBUFFER;
  PROCEDURE FORMFEED;
  PROCEDURE PRINTPIC;
  PROCEDURE COLDSTART;
  PROCEDURE WARMSTART;
```

```
IMPLEMENTATION

  PROCEDURE ROMENABLE;
   CONST ROMSOFF= -12289;
         ROMON= -16128;
   TYPE WINDOW= PACKED ARRAY [0..0] OF 0..255;
   VAR ADDR: INTEGER;
       P: ^WINDOW;
    BEGIN
     ADDR:=ROMSOFF;
     MOVELEFT(ADDR,P,2);
     P^[0]:=0;
     ADDR:=ROMON;
     MOVELEFT(ADDR,P,2);
     P^[0]:=0
   END;

 PROCEDURE SETBYTEVALUE;
   TYPE WINDOW= PACKED ARRAY [0..0] OF 0..255;
   VAR ADDR: INTEGER;
       P: ^WINDOW;
   BEGIN
     ROMENABLE;
     ADDR:=LOC;
     MOVELEFT(ADDR,P,2);
     P^[0]:=VALUE
   END;

 FUNCTION BYTEVALUE;
   TYPE WINDOW= PACKED ARRAY [0..0] OF 0..255;
    VAR ADDR: INTEGER;
       P: ^WINDOW;
   BEGIN
     ROMENABLE;
     ADDR:=LOC;
     MOVELEFT(ADDR,P,2);
     BYTEVALUE:=P^[0]
   END;

  PROCEDURE SEND;
   CONST PRINTUNIT= 6;
   BEGIN
     UNITWRITE(PRINTUNIT,CH,1,0,12)
   END;

 PROCEDURE SETUNIDIRECT;
   CONST MAXBYTE= 255;
         DIRECTION= -12529;
   BEGIN
     SETBYTEVALUE(DIRECTION,MAXBYTE)
   END;
```

```
PROCEDURE SETBIDIRECT;
  CONST MINBYTE= 0;
        DIRECTION= -12529;
  BEGIN
    SETBYTEVALUE(DIRECTION,MINBYTE)
  END;

 FUNCTION UNIDIRECT;
   CONST MAXBYTE= 255;
         MINBYTE= 0;
         DIRECTION= -12529;
  BEGIN
    CASE BYTEVALUE(DIRECTION) OF
      MINBYTE: UNIDIRECT:=FALSE;
      MAXBYTE: UNIDIRECT:=TRUE
    END
  END;

PROCEDURE SETNEGATIVE;
  CONST MINBYTE= 0;
        FLIP= -12524;
  BEGIN
    SETBYTEVALUE(FLIP,MINBYTE)
  END;

PROCEDURE SETPOSITIVE;
  CONST MAXBYTE= 255;
        FLIP= -12524;
  BEGIN
    SETBYTEVALUE(FLIP,MAXBYTE)
  END;

FUNCTION NEGATIVE;
  CONST MAXBYTE= 255;
        MINBYTE= 0;
        FLIP= -12524;
  BEGIN
    CASE BYTEVALUE(FLIP) OF
      MINBYTE: NEGATIVE:=TRUE;
      MAXBYTE: NEGATIVE:=FALSE
    END
  END;

PROCEDURE SETPAGE;
  CONST GRAPHPAGE= -12525;
  BEGIN
    IF PAGE=2 THEN PAGE:=64
      ELSE PAGE:=32;
    SETBYTEVALUE(GRAPHPAGE,PAGE)
  END;
```

```
PROCEDURE SETDARK;
  CONST INTEN= -12528;
  BEGIN
    IF DARKNESS<0 THEN DARKNESS:=0;
    IF DARKNESS>7 THEN DARKNESS:=7;
    SETBYTEVALUE(INTEN,DARKNESS)
  END;

FUNCTION DARK;
  CONST INTEN= -12528;
  BEGIN
    DARK:=BYTEVALUE(INTEN)
  END;

PROCEDURE SETFORM;
  CONST FORMLENGTH= -12531;
  BEGIN
    IF LENGTH<0 THEN LENGTH:=0;
    IF LENGTH>255 THEN LENGTH:=255;
    SETBYTEVALUE(FORMLENGTH,LENGTH)
  END;

FUNCTION FORM;
  CONST FORMLENGTH= -12531;
  BEGIN
    FORM:=BYTEVALUE(FORMLENGTH)
  END;

PROCEDURE SETSPACE;
  CONST INCR= -12530;
  BEGIN
    IF LENGTH<1 THEN LENGTH:=1;
    IF LENGTH>252 THEN LENGTH :=252;
    SETBYTEVALUE(INCR,LENGTH)
  END;

FUNCTION SPACE;
  CONST INCR= -12530;
  BEGIN
    SPACE:=BYTEVALUE(INCR)
  END;

FUNCTION LEFTMARGIN;
  CONST LMAR= -12527;
  BEGIN
    LEFTMARGIN:=BYTEVALUE(LMAR)
  END;
```

```
 FUNCTION RIGHTMARGIN;
  CONST RMAR= -12526;
  BEGIN
    RIGHTMARGIN:=BYTEVALUE(RMAR)
  END;

PROCEDURE SETLEFTMARGIN;
  CONST LMAR= -12527;
  BEGIN
    IF POSITION>=RIGHTMARGIN THEN POSITION:=RIGHTMARGIN-1;
    IF POSITION<0 THEN POSITION:=0;
    SETBYTEVALUE(LMAR,POSITION)
  END;

PROCEDURE SETRIGHTMARGIN;
  CONST RMAR= -12526;
  BEGIN
    IF POSITION<=LEFTMARGIN THEN POSITION:=LEFTMARGIN+1;
    IF POSITION>83 THEN POSITION:=83;
    SETBYTEVALUE(RMAR,POSITION)
  END;

PROCEDURE PRINTBUFFER;
  CONST CF= 6;  (* ASCII FOR CONTROL-F *)
  BEGIN
    SEND(CHR(CF))
  END;

PROCEDURE CLEARBUFFER;
  CONST PRINTUNIT= 6;
  BEGIN
    UNITCLEAR(PRINTUNIT)
  END;

PROCEDURE FORMFEED;
  CONST FF= 12;  (* ASCII FOR FORM-FEED *)
  BEGIN
    SEND(CHR(FF))
  END;

PROCEDURE PRINTPIC;
  CONST CQ= 17;  (* ASCII FOR CONTROL-Q *)
  VAR BIDIRECT: BOOLEAN;
  BEGIN
    IF UNIDIRECT=TRUE THEN BIDIRECT:=FALSE
      ELSE BIDIRECT:=TRUE;
    IF BIDIRECT THEN SETUNIDIRECT;
    SEND(CHR(CQ));
    IF BIDIRECT THEN SETBIDIRECT
  END;
```

```
  PROCEDURE COLDSTART;
    CONST MINBYTE= 0;
          DEFAULT= -12506;
    BEGIN
      SETBYTEVALUE(DEFAULT,MINBYTE)
    END;

  PROCEDURE WARMSTART;
    BEGIN
      SETBIDIRECT;
      SETPOSITIVE;
      SETPAGE(1);
      SETDARK(5);
      SETFORM(40);
      SETSPACE(2);
      SETLEFTMARGIN(2);
      SETRIGHTMARGIN(81);
    END;

BEGIN  (* INITIALIZATION SECTION *)
   COLDSTART;
END.
```

PASCAL OPERAND FORMATS

An useful and often important piece of information concerns the internal structure of Pascal variables. This information can be very useful when sending data (especially complex data formats such as strings) to an assembly routine from a Pascal host program. This article describes a few of the more commonly used variable types. For a complete description of the more complex variables, including records and arrays, see pages 227-228 of the Apple Pascal Operating System Reference Manual.

Machine language (assembly) routines are commonly used when speed is critical, and when the code must access other assembly routines such as PROMs or I/O drivers which can't be re-assembled as part of the program. Also, bit manipulations such as right-shift are much easier to do in assembly than in Pascal.

In the UCSD Pascal system, it's a fairly simple matter to create short assembly programs which can be linked into a Pascal host program. In some cases, it may be sufficient to merely call the assembly routine; however, most routines require data in order to be useful. The means by which data is passed to or from these routines is called a "parameter".

A parameter is a temporary variable created by Pascal for the purpose of passing data to or from a subroutine. The term "formal parameter" implies that the address of the actual variable is passed to the subroutine as a parameter instead of its value.

Certain types of variables may be passed by value, but any variable may be passed by name by simply declaring it to be a formal parameter (a VAR). Pascal does not allow parameters of variable length (with the exception of certain sets and long integers) to be passed on the CPU stack, since this could exceed the stack capacity and crash the operating system, so these parameters are automatically used as formal parameters. A good explanation of the various ways of passing parameters may be found in Peter Grogono's book, "Programming in Pascal".

Before delving into the details, let's define some terms and conventions and which we'll use later on:

BIT = a binary digit (0 or 1). A bit is the smallest unit of information which can be stored in a computer.
NYBBLE = 4 bits (half a byte). A hexadecimal digit is one nybble (pronounced "nibble").
BYTE = 8 bits (2 nybbles). This is the unit of storage which the 6502 processor uses.
WORD = 2 bytes (16 bits). A word is the unit of information which Pascal uses.
LSB = least significant bit
MSB = most significant bit

```
     decimal 65535                                    0
  hexadecimal FFFF <-----------memory-----------> 0000 addresses
               MSB                                LSB
```

This diagram of memory structure will be used in describing the variable formats. Usually, when you write down a number, you write it from left to right. However, Pascal reads data from memory from right to left starting at the least significant byte.

INTEGERS

Integers, in UCSD Pascal, are whole numbers in the range of -32768 to +32767. They are stored in one word (2 bytes). Negative integers are represented in "two's complement", which means that they appear to have positive values (> 32767). By subtracting this positive value from 65536, the negative integer is revealed. Similarly, large positive integers are stored as a complementary negative numbers (remember Integer Basic?). The sign bit (MSB) is 0 if positive, 1 if negative.

```
     <--------byte----------->    <-------byte--------->
    15    14  .  .  .  .  .  8    7  .  .  .  .  .  .  0   <== 16 bits
   SIGN            INTEGER VALUE
```

Example: the number 3 is represented in binary as:

```
  MSB                                                LSB
    0  0  0  0  0  0  0  0    0  0  0  0  0  0  1  1
```

However, -3 shows up as

```
  MSB                                                LSB
    1  1  1  1  1  1  1  1    1  1  1  1  1  1  0  1
```

which also reads as 65533 (or 65536-3)!

Integers may be passed by value or as formal parameters.

REALS

Real number, in UCSD Pascal, are floating point numbers in the range of +/- 1.17550E-38 to +/-3.40282E+38. Real numbers use four bytes (2 words). The binary representation is similar to the proposed IEEE standard for floating point numbers:

```
  31   30 . . . . . . 23  22 . . . . . . . . . . 0  <== 32 bits
 SIGN      EXPONENT              MANTISSA
```

"Mantissa" is the name given to the decimal portion of a number which is expressed in scientific (exponential) notation. The "exponent" indicates the power to which the mantissa is raised. The exponent is represented in base 2 (2^n). In decimal, the number 3 x 10^2 can be seen as a mantissa of 3, an exponent of 2, in base 10 (decimal).

The sign bit refers to the sign of the mantissa, and is 0 if positive, 1 if negative. The exponent is "offset" by 127; that is, a value of 127 in the exponent field corresponds to an exponent of 0. Similarly, if the value is 1, the exponent is -126, and if the field is 254, the exponent is +127. A value of 0 indicates that the real number is 0.

The mantissa of the real number is stored in normalized format in bits 0-22. "Normalizing" a number means adjusting it so that the highest bit is significant (a 1). The exponent indicates how many times (and in which direction) the value was shifted during normalization.

Notice that the MSB of the mantissa of any non-zero number which has been normalized is always a one. The number zero can be treated as a special case by simply setting the exponent to zero. So, to gain additional precision, the mantissa has an implied "1" which is not stored, resulting in a functional 24-bit mantissa, even though only 23 bits are actually used. This gives slightly more than 6 decimal places (single precision) accuracy.

To make this clearer, let's look at some examples:

```
Real number = 1
   MSB 0     01111111      00000000000000000000000 LSB
  Exponent = 127 (2^0)  Mantissa = 1 (the implied 1 isn't stored)

Real number = -9.9
   MSB 1     10000010      00111100110011001100110 LSB
  Exponent = 130 (2^3)  Mantissa = 99000015
```

In the second example, the real number (in binary) appears as 1001.1110011 etc.. During normalization, the decimal point is moved to the left 3 times (incrementing the exponent), and the most significant bit becomes implied. The sign bit is 1, indicating that the number is negative.

Real numbers may be passed by value, or may be defined as formal parameters and passed by address.

CHARACTERS

Characters, by ASCII definition, are simply integers in the range of 0 to 255. Characters take up one word of storage. The ASCII value of the character is stored in the least significant byte. The most significant byte is not used by Pascal and should be ignored.

```
      15 . . . . . . . 8   7 . . . . . . . 0  <== 16 bits
             unused              ASCII
                                  code
```

EXAMPLE: the character "A" has an ASCII value of 65 (hexadecimal 41). Represented in binary, this would be:

```
    MSB x  x  x  x  x  x  x  x   0  1  0  0  0  0  0  1 LSB
     <------ not used-------->      4   (hex)    1
```

Characters can be passed as either actual parameters (pass by value) or formal parameters (pass by address).

STRINGS

A string is a packed array of characters which can be from one to 256 bytes long. The first byte of a string always contains a number from 0 to 255 which indicates the length of the string. One character is stored per byte, and the string ends on a word boundary; that is, if the last character in the string is the first byte of a new word, the other byte of the word is also reserved, but is not used by the string.

Each character of the string can be accessed in a packed array of characters; however, you cannot access the length byte (the 0th element). Doing so will promptly generate the message: "Value Range Error".

EXAMPLE: The string "ABCD" would look like this:

```
        S[4]        S[3]        S[2]        S[1]        S[0]
MSB 01000100    01000011    01000010    01000001    00000100 LSB
      "D"         "C"         "B"         "A"          4
```

Pascal always passes strings by address, since the length may vary.

POINTERS

Address pointers are UNSIGNED integers which occupy 1 word of storage. The format is the same as for integers, except that the values range from 0 to 65535.

EXAMPLE: The address of AN0 (one of the annunciator ports) is hex C058 (49240 decimal). This would be stored as:

```
MSB 1  1  0  0  0  0  0  0   0  1  0  1  1  0  0  0 LSB
    <--------> <--------> <--------> <-------->
        C          0          5          8
```

Pointers, like integers, may be passed by value or by name (formal parameter).

LONG INTEGERS

Long integers are a special type of variable which was first defined at UCSD as part of their extensions to the Pascal language. They are primarily used to handle calculations involving numbers which cannot be represented accurately in floating point (real) format and are too large to store in integer format.

Long integers are stored in BCD (binary coded decimal), one digit per nybble. One entire word is reserved for the sign of the long integer, and the variable must end on a word boundary. Four digits can be contained in one word, so the smallest definable long integer takes up two words of memory. The numbers are padded with leading zeroes when necessary to fill up the last word. The sign will be 0 if positive and 255 if negative (one byte is used).

To illustrate this, let's take a specific example: the long integer -123456 will take 3 words: one for the sign, and two for the digits, since they are stored in multiples of 4. The format will look like this:

```
     <-------- each digit is one nybble --------->
MSB 6   5   4   3   2   1   0   0   0   0   F   F LSB
    <-- word --> <-- word --> < sign word >
```

Long integers should always be passed by address because they have a length which depends on their definition.

BOOLEANS

The Boolean, or binary, variable can have two values: TRUE and FALSE. This is most commonly used in determining yes/no conditions such as equality or set inclusion. This variable is stored in one word, although only the LSB (least significant bit) is used. TRUE is indicated by a 1, and FALSE shows as a 0.

```
    MSB  15 . . . . . . 8  7 . . . . . . 0  LSB
                                          boolean
```

Booleans are most efficient in packed arrays, where each bit of the word is utilized. DRAWBLOCK is probably the best-known example of this use. For an excellent example of the use of boolean packed arrays, look at the program GRAFDEMO on the Apple Pacal diskette APPLE3.

Boolean variables may be passed by value or by address.

OTHER TYPES

In addition to the previously mentioned standard types, Pascal allows the programmer to define a wide variety of non-standard variable types. Probably the most popular example of this is the set.

A set is an arbitrary collection of elements, where each element is assigned an ordinal position (that is, represented by a number). Each element of the set is represented by a name which can be any word of your own choosing (except for Pascal reserved words or other variable definitions already in use). Each name is then associated with one bit in the data definition beginning with bit 0. The set is stored in memory as a series of bits which are identified by the ordinal position of the element in the type definition. A set must end on a word boundary, so, for example, 17 elements would take up 2 words, even though only one bit of the second word is actually used.

EXAMPLE:

```
TYPE COLOR=RED,GREEN,BLUE,YELLOW,BLACK,WHITE
```

is a set of colors. Red occupies position 0, and white is position 5.

```
      <-------------------one word------------------>
 MSB  0  0  0  0  0  0  0  0   0  0  1  1  1  1  1  1  LSB
                                  W  B  Y  B  G  R
                                  h  l  e  l  r  e
                                  i  a  l  u  e  d
                                  t  c  l  e  e
                                  e  k  o     n
                                        w
```

Sets may be passed either by name or by value, with certain restrictions. See page 203 of the Pascal reference manual for details.

In general, complex record types consist of one or more standard types which are stored as described. For the last word on Pascal data types, read Niklaus Wirth's Report in "User Manual and Report" by Jensen and Wirth.

REFERENCES

Apple PASCAL Reference Manual, by Apple Computer Inc. 1979.

Programming in Pascal, by Peter Grogono, Addison Wesley, 1978.

User Manual and Report, by Kathleen Jensen and Niklaus Wirth, Springer-Verlag, 1974.

EXTERNAL TERMINAL SETUP IN PASCAL

Pascal supports a variety of external terminals which can be used to display 80 character lines and upper/lower case. Pages 199-213 of the Apple Pascal Operating System reference manual describe the procedure for reconfiguring the Pascal system to communicate with the external terminal. You must bind the appropriate GOTO routine into SYSTEM.PASCAL using APPLE3:BINDER and use APPLE3:SETUP to tailor SYSTEM.MISCINFO for the specific terminal parameters.

EXTERNAL TERMINALS WITH APPLE SERIAL CARD

The terminal may be connected to the Pascal system using the High-Speed Serial Card in place of the Communications Card. If the serial card is used, the eighth data bit (most significant bit) switch in the terminal must be set to 0. Refer to the operating manual or contact the manufacturer for the location of this switch. With the serial card, all type-ahead is eliminated.

SETUP PARAMETERS

The SETUP program included on the Pascal diskette APPLE3 allows you to reconfigure the SYSTEM.MISCINFO file for a specific external terminal. These parameters are explained in the following sections, arranged by their general function.

SCREEN CONTROL FUNCTIONS

BACKSPACE - sent to screen to move cursor left one space without altering the text. Normal setting for Apple II: CTRL-H (BS).

ERASE LINE - sent to screen to erase entire line where cursor is positioned. Not used in Apple II and should be set to NUL (ASCII 0).

ERASE SCREEN - erases screen and places cursor at "home" position, the upper left corner of the screen. Normal setting: CTRL-L (FF).

ERASE TO END OF LINE - sent to screen to clear from cursor position to end of line. Normal setting: CTRL-] (GS).

ERASE TO END OF SCREEN - clears screen from cursor position to end of screen. Normal setting: CTRL-K (VT).

HAS RANDOM CURSOR ADDRESSING - TRUE for video terminals only (TRUE for Apple video). Enables use of "GOTOXY" sequence.

LEAD-IN TO SCREEN - prefix character for dual-character screen control functions, if any. Set to NUL (ASCII 0) if no lead-in is being used.

MOVE CURSOR HOME - sent to screen to home cursor to "home" position in upper left corner. Normal setting: CTRL-Y (EM).

MOVE CURSOR RIGHT - moves cursor right one space. Normal setting: CTRL-\ (FS).

MOVE CURSOR UP - moves cursor up one line. Normal setting: CTRL-_ (US).

SCREEN HEIGHT - number of lines displayed on terminal. Set to 0 for hard-copy terminal or where paging is not appropriate. Setting is 24 (decimal) for Apple II.

SCREEN WIDTH - number of characters displayed on each line. Usually set to 80 for most terminals, 79 will allow nearly all of the Pascal window to be displayed while generating shortened prompt lines and messages which are better suited for a 40-character line.

KEYBOARD FUNCTIONS

HAS LOWER CASE - FALSE for Apple II keyboard. Set to TRUE if external terminal can generate lower case from its keyboard.

KEY FOR BREAK - character typed to send BREAK to system. Not used in Apple II and should be set to NUL (ASCII 0).

KEY FOR FLUSH - character typed to cancel console output. Processing continues without output until FLUSH character typed again, or until file "KEYBOARD" is accessed. Normal setting: CTRL-F (ACK).

KEY FOR STOP - stops output processing. This is useful when information is streaming out to the console faster than you can read. Output resumes when STOP is toggled again. Normal setting: CTRL-S (DC3).

KEY TO DELETE CHARACTER - causes one character to be removed from current line until nothing is left on line. Normal setting: CTRL-H (BS).

KEY TO DELETE LINE - cancels current line of input. Normal setting: CTRL-X (CAN).

KEY TO END FILE - console EOF character. When reading from "KEYBOARD" or "INPUT", this character will set EOF=TRUE. Normal setting: CTRL-C (ETX).

LEAD-IN FROM KEYBOARD - prefix character for dual-character sequences (if any) generated on keyboard. Set to NUL (ASCII 0) if no lead-in is being used.

The following keyboard functions determine cursor movement, and are usually set to the characters or sequences generated by the arrow keys (cursor keys) on the terminal.

KEY TO MOVE CURSOR DOWN - character typed to move cursor down one line. Apple keyboard setting: CTRL-L (FF).

KEY TO MOVE CURSOR LEFT - character typed to move cursor left one space. Apple keyboard setting: CTRL-H (BS).

KEY TO MOVE CURSOR RIGHT - character typed to move cursor right one space. Apple keyboard setting: CTRL-U (NAK).

KEY TO MOVE CURSOR UP - character typed to move cursor up one line. Apple keyboard setting: CTRL-O (SI).

EDITOR FUNCTIONS

EDITOR ACCEPT KEY - character typed to confirm last action, e. g., accept text inserted, deleted, exchanged, etc. Normal setting: CTRL-C (ETX).

EDITOR ESCAPE KEY - character typed to instruct editor to ignore (escape) all actions performed since last "accept". Normal setting: ESCAPE (ESC).

NON-PRINTING CHARACTER - symbol displayed on screen to indicate presence of non-printable character (such as control characters). Normal setting: "?".

HARDWARE FUNCTIONS

HAS 8510A - always FALSE for Apple II.

HAS BYTE-FLIPPED MACHINE - always FALSE for the Apple II. The Apple II reads data low-byte first. Byte-flipped machines read data high-byte first. (NOTE: This parameter is not included in the SETUP program.)

HAS CLOCK - always FALSE for Apple II. No provision for an accessory real-time clock has been made in the operating system.

OPERATING SYSTEM

HAS SLOW TERMINAL - should be TRUE for terminal running at 600 baud or less. This will issue shortened prompts and messages. FALSE for Apple II.

STUDENT - instructs operating system to simplify operation for the novice. An example would be a default entry into the Editor if a compile error is encountered. May also disallow entry into certain portions of the system. Should be set to FALSE for Apple II.

VERTICAL MOVE DELAY - number of nulls (ASCII 0) to be sent after each carriage return, ERASE TO END OF LINE, ERASE TO END OF SCREEN to allow cursor to complete movement before additional characters are sent. Set to NUL (ASCII 0) for Apple II.

PREFIXED

The following parameters indicate whether or not the control function is to be prefixed by the lead-in character. Set to FALSE if no lead-in is required (as in the case of control characters), and TRUE if the lead-in is needed (ESC sequences, for example).

PREFIXED [DELETE CHARACTER]
PREFIXED [EDITOR ACCEPT KEY]
PREFIXED [EDITOR ESCAPE KEY]
PREFIXED [ERASE LINE]
PREFIXED [ERASE SCREEN]
PREFIXED [ERASE TO END OF LINE]
PREFIXED [ERASE TO END OF SCREEN]
PREFIXED [KEY FOR BREAK]
PREFIXED [KEY FOR FLUSH]
PREFIXED [KEY TO MOVE CURSOR DOWN]
PREFIXED [KEY TO MOVE CURSOR LEFT]
PREFIXED [KEY TO MOVE CURSOR RIGHT]
PREFIXED [KEY TO MOVE CURSOR UP]
PREFIXED [KEY FOR STOP]
PREFIXED [KEY TO DELETE CHARACTER]
PREFIXED [KEY TO DELETE LINE]
PREFIXED [KEY TO END FILE]
PREFIXED [MOVE CURSOR HOME]
PREFIXED [MOVE CURSOR RIGHT]
PREFIXED [MOVE CURSOR UP]
PREFIXED [NON PRINTING CHARACTER]

The program SETUP is located on the diskette APPLE3. When executed, it will read in the current SYSTEM.MISCINFO. You may change a single parameter, or edit the entire file. To edit all the parameters, type "C" (Change), and then "P" (Prompted). If you need instructions, type "H" (Help), or "T" (Teach). These instructions are detailed on pages 199-213 of the Pascal Operating System Reference Manual.

To leave editing mode prior to the end of the file, type "!" (Abort). Then "Q", "Q" (to get to the exit/update level). To leave without updating, simply type "E". If you wish to test your setup, but not save the file on disk, type "M", then "E". To make a permanent record of your changes, type "D", then "E". A file named NEW.MISCINFO will be placed on your boot disk.

If you wish to edit a single parameter, type "C", then "S" (Single). Type the exact name of the field you wish to edit, exactly as it appears below.

Backspace	PREFIXED[Delete Character]
Editor "Accept" Key	PREFIXED[Editor "Accept" Key]
Editor "Escape" Key	PREFIXED[Editor ´Escape´ Key] (this is different!)
Erase Line	PREFIXED[Erase Line]
Erase Screen	PREFIXED[Erase Screen]
Erase to End of Line	PREFIXED[Erase to End of Line]
Erase to End of Screen	PREFIXED[Erase to End of Screen]
Has 8510A	PREFIXED[Key for Break]
Has Clock	PREFIXED[Key for Flush]
Has Lower Case	PREFIXED[Key for Moving Cursor Down]
Has Random Cursor Addr.	PREFIXED[Key for Moving Cursor Left]
Has Slow Terminal	PREFIXED[Key for Moving Cursor Right]
Key for Break	PREFIXED[Key for Moving Cursor Up]
Key for Flush	PREFIXED[Key for Stop]
Key for Stop	PREFIXED[Key to Delete Character]
Key to Delete Character	PREFIXED[Key to Delete Line]
Key to Delete Line	PREFIXED[Key to End File]
Key to End File	PREFIXED[Move Cursor Home]
Key to Move Cursor Down	PREFIXED[Move Cursor Right]
Key to Move Cursor Left	PREFIXED[Move Cursor Up]
Key to Move Cursor Right	PREFIXED[Non-printing Character]
Key to Move Cursor Up	Screen Height
Lead-in from Keyboard	Screen Width
Lead-in to Screen	Student
Move Cursor Home	Vertical Move Delay
Move Cursor Right	
Move Cursor Up	
Non-printing Character	

APPLE STANDARD VIDEO AND KEYBOARD

GOTO CODE: The system is already configured for the Apple keyboard and video. No modifications are necessary. These are the "normal" settings suggested in the preceding section.

SETUP PARAMETERS:

			PREFIXED:	
Backspace	CTRL-H	(08)	[Delete Character]	FALSE
Editor "Accept" Key	CTRL-C	(03)	[Editor "Accept" Key]	FALSE
Editor "Escape" Key	ESC	(27)	[Editor ´Escape´ Key]	FALSE
Erase Line	NUL	(00)	[Erase Line]	FALSE
Erase Screen	CTRL-L	(12)	[Erase Screen]	FALSE
Erase to End of Line	CTRL-]	(29)	[Erase to End of Line]	FALSE
Erase to End of Screen	CTRL-K	(11)	[Erase to End of Screen]	FALSE
Has 8510A	FALSE		[Key for Break]	FALSE
Has Clock	FALSE		[Key for Flush]	FALSE
Has Lower Case	FALSE		[Key for Moving Cursor Down]	FALSE
Has Random Cursor Addr.	TRUE		[Key for Moving Cursor Left]	FALSE
Has Slow Terminal	FALSE		[Key for Moving Cursor Right]	FALSE
Key for Break	NUL	(00)	[Key for Moving Cursor Up]	FALSE
Key for Flush	CTRL-F	(06)	[Key for Stop]	FALSE
Key for Stop	CTRL-S	(19)	[Key to Delete Character]	FALSE
Key to Delete Character	CTRL-H	(08)	[Key to Delete Line]	FALSE
Key to Delete Line	CTRL-X	(24)	[Key to End File]	FALSE
Key to End File	CTRL-C	(03)	[Move Cursor Home]	FALSE
*Key to Move Cursor Down	CTRL-L	(12)	[Move Cursor Right]	FALSE
*Key to Move Cursor Left	CTRL-H	(08)	[Move Cursor Up]	FALSE
*Key to Move Cursor Right	CTRL-U	(21)	[Non-printing Character]	FALSE
*Key to Move Cursor Up	CTRL-O	(15)	Screen Height	24
Lead-in from Keyboard	NUL	(00)	Screen Width	80
Lead-in to Screen	NUL	(00)	Student	FALSE
Move Cursor Home	CTRL-Y	(25)	Vertical Move Delay	0
Move Cursor Right	CTRL-\	(28)		
Move Cursor Up	CTRL-_	(31)	* These may be altered by the user for convenience.	
Non-printing Character	"?"	(63)		

SOROC IQ120

GOTO CODE: use APPLE3:SOROC.GOTO No modifications are necessary.

SETUP PARAMETERS:

			PREFIXED:	
Backspace	CTRL-H	(08)	[Delete Character]	FALSE
Editor "Accept" Key	CTRL-C	(03)	[Editor "Accept" Key]	FALSE
Editor "Escape" Key	ESC	(27)	[Editor ´Escape´ Key]	FALSE
Erase Line	NUL	(00)	[Erase Line]	FALSE
Erase Screen	"*"	(42)	[Erase Screen]	TRUE
Erase to End of Line	"T"	(84)	[Erase to End of Line]	TRUE
Erase to End of Screen	"Y"	(89)	[Erase to End of Screen]	TRUE
Has 8510A	FALSE		[Key for Break]	FALSE
Has Clock	FALSE		[Key for Flush]	FALSE
Has Lower Case	TRUE		[Key for Moving Cursor Down]	FALSE
Has Random Cursor Addr.	TRUE		[Key for Moving Cursor Left]	FALSE
Has Slow Terminal	FALSE		[Key for Moving Cursor Right]	FALSE
Key for Break	NUL	(00)	[Key for Moving Cursor Up]	FALSE
Key for Flush	CTRL-F	(06)	[Key for Stop]	FALSE
Key for Stop	CTRL-S	(19)	[Key to Delete Character]	FALSE
Key to Delete Character	CTRL-H	(08)	[Key to Delete Line]	FALSE
Key to Delete Line	DEL	(127)	[Key to End File]	FALSE
Key to End File	CTRL-C	(03)	[Move Cursor Home]	FALSE
*Key to Move Cursor Down	CTRL-J	(10)	[Move Cursor Right]	FALSE
*Key to Move Cursor Left	CTRL-H	(08)	[Move Cursor Up]	FALSE
*Key to Move Cursor Right	CTRL-L	(12)	[Non-printing Character]	FALSE
*Key to Move Cursor Up	CTRL-K	(11)	Screen Height	24
Lead-in from Keyboard	NUL	(00)	Screen Width	80
Lead-in to Screen	ESC	(27)	Student	FALSE
Move Cursor Home	CTRL-^	(30)	Vertical Move Delay	0
Move Cursor Right	CTRL-L	(12)		
Move Cursor Up	CTRL-K	(11)	* These may be altered by the user for convenience.	
Non-printing Character	"?"	(63)		

HAZELTINE 1500

GOTO CODE: use APPLE3:HAZEL.GOTO No modifications are necessary.

SETUP PARAMETERS:

		PREFIXED:	
Backspace	CTRL-H (08)	[Delete Character]	FALSE
Editor "Accept" Key	CTRL-C (03)	[Editor "Accept" Key]	FALSE
Editor "Escape" Key	ESC (27)	[Editor ´Escape´ Key]	FALSE
Erase Line	NUL (00)	[Erase Line]	FALSE
Erase Screen	CTRL-\ (28)	[Erase Screen]	TRUE
Erase to End of Line	CTRL-O (15)	[Erase to End of Line]	TRUE
Erase to End of Screen	CTRL-X (24)	[Erase to End of Screen]	TRUE
Has 8510A	FALSE	[Key for Break]	FALSE
Has Clock	FALSE	[Key for Flush]	FALSE
Has Lower Case	TRUE	[Key for Moving Cursor Down]	FALSE
Has Random Cursor Addr.	TRUE	[Key for Moving Cursor Left]	FALSE
Has Slow Terminal	FALSE	[Key for Moving Cursor Right]	FALSE
Key for Break	NUL (00)	[Key for Moving Cursor Up]	FALSE
Key for Flush	CTRL-F (06)	[Key for Stop]	FALSE
Key for Stop	CTRL-S (19)	[Key to Delete Character]	FALSE
Key to Delete Character	CTRL-H (08)	[Key to Delete Line]	FALSE
Key to Delete Line	DEL (127)	[Key to End File]	FALSE
Key to End File	CTRL-C (03)	[Move Cursor Home]	TRUE
*Key to Move Cursor Down	CTRL-K (11)	[Move Cursor Right]	FALSE
*Key to Move Cursor Left	CTRL-H (08)	[Move Cursor Up]	TRUE
*Key to Move Cursor Right	CTRL-P (16)	[Non-printing Character]	FALSE
*Key to Move Cursor Up	CTRL-L (12)	Screen Height	24
Lead-in from Keyboard	NUL (00)	Screen Width	80
Lead-in to Screen	"~" (126)	Student	FALSE
Move Cursor Home	CTRL-R (18)	Vertical Move Delay	0
Move Cursor Right	CTRL-P (16)		
Move Cursor Up	CTRL-L (12)	* These may be altered by the user for convenience.	
Non-printing Character	"?" (63)		

ADDITIONAL NOTES: The four parity switches must be set to OFF. The CR/Auto LF switch must be set to CR. These DIP switches are located beneath the front panel plate of the terminal.

HAZELTINE 1510

GOTO CODE: use APPLE3:HAZEL.GOTO No modifications are necessary.

SETUP PARAMETERS:

		PREFIXED:	
Backspace	CTRL-H (08)	[Delete Character]	FALSE
Editor "Accept" Key	CTRL-C (03)	[Editor "Accept" Key]	FALSE
Editor "Escape" Key	ESC (27)	[Editor ´Escape´ Key]	FALSE
Erase Line	NUL (00)	[Erase Line]	FALSE
Erase Screen	CTRL-\ (28)	[Erase Screen]	TRUE
Erase to End of Line	CTRL-O (15)	[Erase to End of Line]	TRUE
Erase to End of Screen	CTRL-X (24)	[Erase to End of Screen]	TRUE
Has 8510A	FALSE	[Key for Break]	FALSE
Has Clock	FALSE	[Key for Flush]	FALSE
Has Lower Case	TRUE	[Key for Moving Cursor Down]	TRUE
Has Random Cursor Addr.	TRUE	[Key for Moving Cursor Left]	FALSE
Has Slow Terminal	FALSE	[Key for Moving Cursor Right]	FALSE
Key for Break	NUL (00)	[Key for Moving Cursor Up]	TRUE
Key for Flush	CTRL-F (06)	[Key for Stop]	FALSE
Key for Stop	CTRL-S (19)	[Key to Delete Character]	FALSE
Key to Delete Character	CTRL-H (08)	[Key to Delete Line]	FALSE
Key to Delete Line	DEL (127)	[Key to End File]	FALSE
Key to End File	CTRL-C (03)	[Move Cursor Home]	TRUE
*Key to Move Cursor Down	CTRL-K (11)	[Move Cursor Right]	FALSE
*Key to Move Cursor Left	CTRL-H (08)	[Move Cursor Up]	TRUE
*Key to Move Cursor Right	CTRL-P (16)	[Non-printing Character]	FALSE
*Key to Move Cursor Up	CTRL-L (12)	Screen Height	24
Lead-in from Keyboard	"~" (126)	Screen Width	80
Lead-in to Screen	"~" (126)	Student	FALSE
Move Cursor Home	CTRL-R (18)	Vertical Move Delay	0
Move Cursor Right	CTRL-P (16)		
Move Cursor Up	CTRL-L (12)	* These may be altered by the user for convenience.	
Non-printing Character	"?" (63)		

IBM 3101-22

GOTO CODE:

```
(*$U-*)
PROGRAM GOXY;  (* For IBM-3010-22 *)
PROCEDURE FGOTOXY(X,Y:INTEGER);
VAR SEND: PACKED ARRAY [0..3] OF 0..255;
BEGIN
  IF X>79 THEN X:=79 ELSE IF X<0 THEN X:=0;
  IF Y>23 THEN Y:=23 ELSE IF Y<0 THEN Y:=0;
  SEND[0]:=27; (* Lead-in *)
  SEND[1]:=ORD('Y');
  SEND[2]:=32+X;
  SEND[3]:=32+Y;
  UNITWRITE(2,SEND,4)
END;

BEGIN (* Dummy Main *)
END.
```

SETUP PARAMETERS:

Backspace	CTRL-H	(08)
Editor "Accept" Key	CTRL-C	(03)
Editor "Escape" Key	CTRL-Q	(17)
Erase Line	"O"	(79)
Erase Screen	"L"	(76)
Erase to End of Line	"I"	(73)
Erase to End of Screen	"J"	(74)
Has 8510A	FALSE	
Has Clock	FALSE	
Has Lower Case	TRUE	
Has Random Cursor Addr.	TRUE	
Has Slow Terminal	FALSE	
Key for Break	NUL	(00)
Key for Flush	CTRL-F	(06)
Key for Stop	CTRL-S	(19)
Key to Delete Character	CTRL-H	(08)
Key to Delete Line	CTRL-X	(24)
Key to End File	CTRL-C	(03)
*Key to Move Cursor Down	"B"	(66)
*Key to Move Cursor Left	"D"	(68)
*Key to Move Cursor Right	"C"	(67)
*Key to Move Cursor Up	"A"	(65)
Lead-in from Keyboard	ESC	(27)
Lead-in to Screen	ESC	(27)
Move Cursor Home	"H"	(72)
Move Cursor Right	"C"	(67)
Move Cursor Up	"A"	(65)
Non-printing Character	"?"	(63)

PREFIXED:	
[Delete Character]	FALSE
[Editor "Accept" Key]	FALSE
[Editor 'Escape' Key]	FALSE
[Erase Line]	TRUE
[Erase Screen]	TRUE
[Erase to End of Line]	TRUE
[Erase to End of Screen]	TRUE
[Key for Break]	FALSE
[Key for Flush]	FALSE
[Key for Moving Cursor Down]	TRUE
[Key for Moving Cursor Left]	TRUE
[Key for Moving Cursor Right]	TRUE
[Key for Moving Cursor Up]	TRUE
[Key for Stop]	FALSE
[Key to Delete Character]	FALSE
[Key to Delete Line]	FALSE
[Key to End File]	FALSE
[Move Cursor Home]	TRUE
[Move Cursor Right]	TRUE
[Move Cursor Up]	TRUE
[Non-printing Character]	FALSE
Screen Height	24
Screen Width	80
Student	FALSE
Vertical Move Delay	0

* These may be altered by the user for convenience.

VC 404

GOTO CODE:

```
(*$U-*)
PROGRAM GOXY;  (* For VC 404 Video Terminal *)
PROCEDURE FGOTOXY(X,Y:INTEGER);
VAR SEND: PACKED ARRAY [0..2] OF 0..255;
BEGIN
  IF X>79 THEN X:=79 ELSE IF X<0 THEN X:=0;
  IF Y>23 THEN Y:=23 ELSE IF Y<0 THEN Y:=0;
  SEND[0]:=16; (* Lead-in and DC1 *)
  SEND[1]:=32+Y;
  SEND[2]:=32+X;
  UNITWRITE(2,SEND,3,0,12)
END;

BEGIN (* DUMMY MAIN *)
END.
```

SETUP PARAMETERS:

			PREFIXED:	
Backspace	CTRL-H	(08)	[Delete Character]	FALSE
Editor "Accept" Key	CTRL-C	(03)	[Editor "Accept" Key]	FALSE
Editor "Escape" Key	ESC	(27)	[Editor ´Escape´ Key]	FALSE
Erase Line	NUL	(00)	[Erase Line]	FALSE
Erase Screen	CTRL-X	(24)	[Erase Screen]	FALSE
Erase to End of Line	CTRL-V	(22)	[Erase to End of Line]	FALSE
Erase to End of Screen	CTRL-W	(23)	[Erase to End of Screen]	FALSE
Has 8510A	FALSE		[Key for Break]	FALSE
Has Clock	FALSE		[Key for Flush]	FALSE
Has Lower Case	TRUE		[Key for Moving Cursor Down]	FALSE
Has Random Cursor Addr.	TRUE		[Key for Moving Cursor Left]	FALSE
Has Slow Terminal	FALSE		[Key for Moving Cursor Right]	FALSE
Key for Break	NUL	(00)	[Key for Moving Cursor Up]	FALSE
Key for Flush	CTRL-F	(06)	[Key for Stop]	FALSE
Key for Stop	CTRL-S	(19)	[Key to Delete Character]	FALSE
Key to Delete Character	CTRL-H	(08)	[Key to Delete Line]	FALSE
Key to Delete Line	CTRL-X	(24)	[Key to End File]	FALSE
Key to End File	CTRL-C	(03)	[Move Cursor Home]	FALSE
*Key to Move Cursor Down	CTRL-J	(10)	[Move Cursor Right]	FALSE
*Key to Move Cursor Left	CTRL-H	(08)	[Move Cursor Up]	FALSE
*Key to Move Cursor Right	CTRL-U	(21)	[Non-printing Character]	FALSE
*Key to Move Cursor Up	CTRL-Z	(26)	Screen Height	24
Lead-in from Keyboard	NUL	(00)	Screen Width	80
Lead-in to Screen	NUL	(00)	Student	FALSE
Move Cursor Home	CTRL-Y	(25)	Vertical Move Delay	10
Move Cursor Right	CTRL-U	(21)		
Move Cursor Up	CTRL-Z	(26)	* These may be altered by the user	
Non-printing Character	"?"	(63)	for convenience.	

PASCAL LANGUAGE REFERENCE MANUAL ERRATA
030-0101-00

Page 22
There is no mention of the CHR (X) function.

Page 86
ORD will accept any ordinal type (integer, character, or user defined type) and return the ordinate of the argument. We should mention that the most common use for ORD is to return the ASCII value of a character.

Page 148
The two step boot cannot use APPLE1: for the first step because RESET does a cold boot in Version 1.1. Using APPLE3: is OK for the first part of the boot.

PASCAL OPERATING SYSTEM MANUAL ERRATA
030-0100-00

Page 105
The first paragraph is a duplicate of the last paragraph on page 104.

Page 107
Maximum Editor file size should be 38 blocks.

Page 130
SYSTEM.SWAPDISK is misspelled.

Page 140
The example for replace should read: /R/<CTRL-L>//<RET>/.

Page 150
".CODE" does not have to be specified for Output file from Linker.

Page 199
The HAZEL.MISCINFO file found on Apple 3: is designed for the Hazeltine 1510 terminal, and will not work completely on the 1500 as set up.

Page 286
Line 9: RESET does a cold boot, not a warm boot, in Version 1.1.

ATTACH-BIOS document for Apple II Pascal 1.1

By Barry Haynes

Jan 12, 1981

This document is intended for Apple II Pascal internal applications writers, vendors and users who need to attach their own drivers to the system or who need more detailed information about the 1.1 BIOS. It is divided into two sections, one explaining how to use the ATTACH utility available through technical support and the other giving general information about the BIOS. It is a good idea to read this whole document before assuming something is missing or hasn't been completely explained. This document is intended for more advanced users who already know a fair amount about I/O devices and how to write device drivers. It is not intended to be a simple step by step description of how to write your first device driver, nor does it claim to be a complete description of all there is to know about the Pascal BIOS.

The Apple Pascal UCSD system has various levels of I/O that are each responsible for different types of actions. It was divided at UCSD into these levels to make it easy to bring up the system on various processors and also various configurations of the same processor and yet have things look the same to the Pascal level regardless of what was below that level. The levels are:

LEVEL AND TYPES OF IO ACTIONS

Pascal

READ & WRITE
BLOCKREAD & BLOCKWRITE
UNITREAD & UNITWRITE
UNITCLEAR
UNITSTATUS

RSP (Runtime Support Package)

This is part of the interpreter and is the middle man between the above types of I/O and the below types of I/O. All the above types are translated by the compiler and operating system into UNITREAD, UNITWRITE, UNITCLEAR and UNITSTATUS if they are not already in that form in the Pascal program. The RSP checks the legality of the parameters passed and reformats these calls into calls to the BIOS routines below. The RSP also expands DLE (blank suppression) characters, adds line feeds to carriage returns, checks for end of file (CTRL-C from CONSOLE:), monitors UNITREAD/UNITWRITE control word commands, makes calls to attached devices if present, echoes to the CONSOLE:.

BIOS (Basic I/O Subsystem)

This are the lowest level device driver routines. This is the level at which you can attach new drivers to replace or work with the regular system drivers and also attach drivers for devices that will be completely defined by you.

I. RECONFIGURING THE BIOS TO ADD YOUR OWN DRIVERS USING THE ATTACH UTILITY.

INTRODUCTION

With the Apple Pascal 1.1 System (both regular and runtime 1.1), there is an automatic method for you to configure your own drivers into the system. This method requires you to write the drivers following certain rules and to use the programs ATTACHUD.CODE and SYSTEM.ATTACH provided through Apple Technical Support. At boot time, the initialization part of SYSTEM.PASCAL looks for the program SYSTEM.ATTACH on the boot drive. If it finds SYSTEM.ATTACH, it executes it before executing SYSTEM.STARTUP. SYSTEM.ATTACH will use the files ATTACH.DATA and ATTACH.DRIVERS which must also be on the boot disk. ATTACH.DATA is a file the developer will make using the program ATTACHUD. It tells SYSTEM.ATTACH the needed information about the drivers it will be attaching. ATTACH.DRIVERS is a file containing all the drivers to be attached and is constructed by the developer using the standard LIBRARY program. The drivers are put on the Pascal Heap below the point that a regular program can access it. They do take away Stack-Heap space (equal to the size of the drivers attached) from that available to Pascal code files but this should not be a problem unless the drivers are very large or the code files very hungry in their use of memory. Since these drivers are configured into the system after the operating system starts to run, this method will not work for configuring drivers for devices that the system must cold boot from. Some of supporting code in the RSP, boot and BIOS may make the task of bringing up boot drivers easier, though. The advantages to this kind of setup are:

1. Software Vendors can use the ATTACHUD program to put their own drivers into the system at boot time. This will be invisible to the user.

2. There can be no problems losing drivers due to improper heap management since the drivers are put on the heap by the operating system and before any user program can allocate heap space.

3. This method does not freeze parts of the system to special memory locations since it enforces the clean methodology of using relocatable drivers.

USING ATTACHUD

ATTACHUD.CODE will ask you questions about the drivers you want to attach to the system. It makes a file called ATTACH.DATA which tells SYSTEM.ATTACH which drivers to attach to the system, what unit numbers to attach them to and other information. The options covered by ATTACHUD are:

1. A driver can be attached to one of the system devices, then all I/O to this device (PRINTER: for example) will go to this new driver. In the case of a new driver for a disk device the user will have to specify which of the 6 standard disk units will go to this new driver. This will allow replacement of standard drivers with custom ones without having to restrict the I/O interface to UNITREAD and UNITWRITE as is the case with option 2.

2. A driver can be attached to one of 16 user devices. I/O to these will be done with UNITREAD and UNITWRITE to device numbers 128-143.

3. A method will be included to allow the attached driver to start on an N-byte boundary. The driver writer will be responsible for aligning his code from that point.

4. More than one unit can be attached to the same driver. This way only one copy of the driver resides in memory and I/O to all the attached units goes to this one driver. It is up to the driver to decide which unit's I/O it is doing. How this is done is explained below.

5. The initialize routine for any attached driver can be called by SYSTEM.ATTACH after it has attached the driver and before any programs can be executed.

6. In case any of your programs use the Hires pages, you can specify in ATTACHUD that drivers must not be put on the heap over these areas. Your drivers would have to be quite large before they could possibly overlap the Hires pages.

Follow through this example of a session with ATTACHUD where the options available are completely described. First execute ATTACHUD:

You will be given the prompt:

```
Apple pascal attachud [1.1]

Enter name of attach data file:
```

This is asking for what you want the output file from this session with ATTACHUD to be called. You could call it ATTACH.DATA or some other name and then rename it to ATTACH.DATA when you put it on the boot disk with SYSTEM.ATTACH.

If you ever get a message of the form:

```
ERROR => some error
Try again (RETURN to exit program):
```

then just retype what was requested on the previous prompt after deciding what mistake you made while typing it the first time.

The next prompt is:

```
These next questions will determine if
attached drivers can reside in the hires
pages.  It will be assumed they can for the
page in question if you answer no to the
prompt for that page.
Will you ever use the (2ØØØ.3FFF hex)
hires page?
```

Followed by:

```
Will you ever use the (4ØØØ.5FFF hex)
hires page?
```

You should answer "Yes" to the question for a particular Hires page if you will ever be running a program that uses that Hires page while the drivers are attached. You don't want the possibility of your driver residing in the Hires page if that page will be clobbered by one of your programs. After answering the Hires questions you will be asked the following questions once for each driver you will be attaching:

```
What is the name of this driver?  This
must be the .PROC name in it's assembly
source (RETURN to exit program):
```

This must be the name of one of the drivers in the ATTACH.DRIVERS that will be used with this ATTACH.DATA. The length of this name must not be more than 8 characters. After entering the name you will be asked:

```
Which unit numbers should refer to this
device driver?

Unit number (RETURN to abort program):
```

You must enter a unit number in the range 1,2,4..12,128..143 or will be given an error message. You cannot attach a character unit (CONSOLE:, PRINTER: or REMOTE:) to the same driver as a block structured unit and if you try you will be given the message:

```
You can't attach a character unit and
a block unit to the same driver.  I
will remove the last unit# you entered.
Type RETURN to continue:
```

If you don't get the above error, you will be asked:

```
Do you want this unit to be
initialized at boot time?
```

A "Yes" response will put the unit number just entered on a list of units that SYSTEM.ATTACH will call UNITCLEAR on after attaching all the drivers. This gives you a way to have the system make an initialize call on your attached unit at boot time. A "No" response will mean that no boot time init call will be made on this unit to the driver you just attached.

You will be eventually asked:

```
Do you want another unit number to refer
to this device driver?:
```

A "Yes" response will get you to the Unit number prompt again and a "No" response will get you to the prompt:

```
Do you want this driver to start on a
certain byte boundary?
```

A "Yes" here will give you more prompts:

```
The boundary can be between Ø and 256.
  Ø=>Driver can start anywhere.(default)
  8=>Driver starts on 8 byte boundary.
  N=>Driver starts on N byte boundary.
256=>Driver starts on 256 byte PAGE boundary.
Enter boundary (RETURN to exit program):
```

And the last line of the prompt will repeat until you enter a boundary in the correct range. The boundary refers to the memory location where the first byte of the driver is loaded. If your driver needs to be aligned on some N-byte boundary you can insure it will be using this mechanism. If you know how the driver's origin is aligned, You can align internal parts of your driver however you want. Finally you will get to the prompt:

```
Do you want to attach another driver?
```

And if you answer "Yes" to this you will return to the 'What is the name of this driver' prompt and answering "No" will end the program, saving the data file you have made.

THE DRIVER

Drivers must be written in assembly using the Pascal Assembler. They must not use the .ABSOLUTE option, so the drivers can be relocated as they are brought in by the system. Each driver must be assembled separately with no external references. When all drivers are assembled, use the LIBRARY program (in the same way you would use it to put units into a library) to put all the drivers in one file. Name this file SYSTEM.DRIVERS. See further explanation of making SYSTEM.DRIVERS below.

A. Considerations for all drivers:

1. Study the examples below as certain information is only documented there.

2. Refer to the Apple II Pascal memory map below and you will see that parts of the interpreter and BIOS reside in the same address range and are bank-switched. The system automatically folds in the BIOS for drivers added using ATTACH. Most of these drivers will have to make calls to CONCK if they want type ahead to continue to work properly. CONCK is the BIOS routine that monitors the keyboard. See the example drivers below to be sure you are doing this correctly. You cannot call CONCK through the CONCK vector at BFØA (see BIOS part of this document) because this call would go through the same mechanism used to get to your driver and the return address to Pascal would be lost.

3. All attached drivers must be written with one common entry point for read, write, init and status. The driver will use the X-Register contents to decide which type of I/O call this is and jump to the appropriate place within its code. The X-Register is decoded as follows:

```
Ø -->read (no bits set)
1 -->write (bit Ø set)
2 -->init (bit 1 set) { The Pascal statement
     UNITCLEAR(UNITNUMBER);  makes an init call for
     unit UNITNUMBER }
4 -->status (bit 2 set)
```

4. The drivers must also pop a return address off the stack, save it and later push it to do a RTS when the driver is finished. All other parameters must be removed from the stack by the driver. For all calls, the return address will be the top word on the stack.

5. SYSTEM.ATTACH will make a copy of the normal system jump vector (the vector after the fold) and put this on the heap. There will be a pointer to this vector at ØE2. Your drivers can use this vector to get to the normal system drivers for device numbers 1..12. See example below.

6. All drivers must pass back a completion code in the X-Register corresponding to the table on page 28Ø of the 1.1 "Apple II Apple Pascal Operating System Reference Manual".

7. In references below to parameters passed on the stack, all parameters are one word parameters so they require two bytes to be popped from the stack by the driver.

8. Control word format for UNITREAD and UNITWRITE

```
bits  15..13      12..6       5      4      3      2      1..Ø
      user        reserved    type B type A nocrlf nospec reserved
      defined     for future  chars  chars                for future
      functions   expansion                               expansion
```

type B =Ø ==>System will check for CTRL-S and CTRL-F from CONSOLE: during the time of this UNITIO call.
=1 ==>System will not check for CTRL-S and CTRL-F during this UNITIO.

type A =Ø ==>If using Apple Keyboard, system will check for CTRL-A, CTRL-Z, CTRL-K, CTRL-W, and CTRL-E from CONSOLE: during the period of this UNITIO.
=1 ==>System will not check for the characters during this UNITIO.

nocrlf =Ø ==>line feeds are added to carriage returns by the Interpreter.
=1 ==>no line feeds are added

nospec =Ø ==>DLE's (blank suppression code) are expanded on output and the EOF character is detected on input
=1 ==>nothing special is done to DLE's on output and EOF on input.

default setting for all control word bits = Ø.

9. Control word format for UNITSTATUS

```
bits  15..13      12..2       1          Ø
      user        reserved    for        direction
      defined     for future  purpose
```

direction =Ø ==>Status of output channel is requested
=1 ==>Status of input channel is requested

purpose =Ø ==>Call is for unit status
=1 ==>Call is for unit control

1Ø. These are the new vectors and routines added to the BIOS to make attach work. The RSP, bootstrap, and readseg were also modified to allow for attaches.

```
UDJMPVEC    ;Jump vector for user devices, offset=Ø to unattached device.
            ;The correct addresses are initialized by SYSTEM.ATTACH
            ;See locations section of BIOS part below for pointers to
            ;this vector.
            JMP     Ø               ;Unit 128
            JMP     Ø               ;Unit 129
            .
            .
            JMP     Ø               ;Unit 143

DISKNUM     ;If high byte=FF then device is not a disk drive
            ;else if high byte=Ø then device is a regular disk drive and
            ;    low byte=drive number
            ;  else driver for this disk drive has been attached by
            ;     SYSTEM.ATTACH and the driver address is stored in this
            ;     word. (Driver address has to be the address-1 for RTS
            ;     in PSUBDR to work correctly, remember this for
            ;     ATTACH.  PSUBDR is listed below.)
            ;See locations section of BIOS part below for pointers to
            ;this vector.
            .WORD   ØFFFF           ;Unit #1
            .WORD   ØFFFF           ;Unit #2  (ATTACH would modify the
            .WORD   ØFFFF           ;Unit #3   words for units 4,5,9..12
            .WORD   Ø               ;Unit #4   if a different disk
            .WORD   1               ;Unit #5   driver were attached to
            .WORD   ØFFFF           ;Unit #6   any of them)
            .WORD   ØFFFF           ;Unit #7
            .WORD   ØFFFF           ;Unit #8
            .WORD   4               ;Unit #9
            .WORD   5               ;Unit #1Ø
            .WORD   2               ;Unit #11
            .WORD   3               ;Unit #12

UDRWIS      ;Routine to get to an attached driver through UDJMPVEC
            ;Assume unit number in A-Register and operation to be
            ;performed in X-Register. See the jump vector in the BIOS
            ;sections to see how you get to this routine.
            STA     TT1
            AND     #7F             ;Clear top bit of unit number
            STA     TT2             ;Make address in UDJMPVEC table
            ASL     A               ;Address=A-Reg*3 + base of table
            CLC
            ADC     TT2             ;Now we have (A-Reg*3).
            ADC     #JVECTRS        ;Add in low byte of base of table
            STA     TT2             ;having no carry problem with only
            LDA     #Ø              ;16 UD's.
            ADC     JVECTRS+1       ;JVECTRS is a word pointing to the
                                    ;base of UDJMPVEC.
            STA     TT2+1
            LDA     TT1
            JMP     @TT2
```

```
PSUBDR      ;Routine to get to an attached driver through DISKNUM
            ;We assume on entry, A-Register=unit number, Y-Register=
            ;DISKNUM offset and X-Register=command to be performed by
            ;the substituted disk driver.
            ;See the jump vector in the BIOS sections to see how you
            ;get to this routine.
            STA     TT1          ;Save unit number.
            LDA     DISKNUM-1,Y  ;Store MSB of driver address.
            PHA
            LDA     DISKNUM-2,Y  ;Store LSB of driver address.
            PHA
            LDA     TT1          ;Restore unit number to A-Register.
            RTS                  ;Jump to substituted driver.  This
                                 ;assumes the driver address in
                                 ;DISKNUM = (ADDRESS OF DRIVER)-1 for
                                 ;the RTS to work
```

B. Special considerations when attaching drivers for the system devices, unit numbers 1..12.

1. Character Oriented Devices (Pass the character to be read-written in the A-Register and make BIOS calls one character at a time from RSP level. On entry, the unit number will be in the Y-Register in case you wanted to attach all character oriented devices to the same driver). If you attach REMOTE: and/or PRINTER: to the same driver as CONSOLE:, all will have their jump vectors pointing to the start of the driver+3 bytes. See further discussion on this below.

   a. Units 1 and 2 (CONSOLE: and SYSTERM:)

      1) These must both go to the same driver.

      2) The system CONCK routine will be patched to jump to the start of the driver. The CONCK routine gets characters entered at the keyboard and fills the type-ahead buffer. See the example CONSOLE: driver below.

      3) Because of item 2, the entry point for normal calls (not CONCK calls) to the attached driver will be 3 bytes beyond the start of the driver.

      4) The interpreter takes care of expanding blank suppression codes (DLE's), echo to the screen, EOF (the end of file character), and adding line feeds to every carriage return. Your driver doesn't need to do this.

5) CONSOLE: read and write have only the return address on the stack. The stack for CONSOLE: init looks like:

```
POINTER TO BREAK VECTOR (This should be stored at
                         location BF16..BF17 by CONSOLE:
                         init.)
POINTER TO SYSCOM       (This should be stored at
                         location F8..F9 by CONSOLE:
                         init.)
                        (Also at init time, the Flush
                         and Start/Stop conditions
                         should be set to normal and
                         the type-ahead queue should
                         be emptied.)
RETURN ADDRESS          <--TOS (top of stack)
```

The stack for CONSOLE: status looks like:

```
POINTER TO STATUS RECORD
CONTROL WORD
RETURN ADDRESS          <--TOS
```

6) A status request should return, in the first word of the status record, the number of characters currently queued in the direction asked for. This is the number of characters in the type-ahead buffer. If no type-ahead is being used then output status should always return a Ø and input status a 1 if a character is waiting to be read, otherwise a Ø.

7) Since we are using 7 bit ASCII codes, the CONSOLE: read routine should zero the high order bit of all characters it reads from the keyboard and passes back to Pascal (to the RSP). The CONSOLE: write routine should transfer all 8 bits as received from the RSP since many devices use 8-bit control codes.

8) The RSP will send both upper and lower case chars to the CONSOLE: write routine. The write routine should map the lower to upper if the device cannot handle lower case.

9) CONSOLE: Output Requirements:

   a) CR (ØD hex): A carriage return should move the cursor to the beginning of the current line.

   b) LF (ØA hex): A line feed should move the cursor to the next line but not change the column position. If the cursor is on the last line on the screen when a line feed is sent, the rest of the screen should scroll up one line and the bottom line be cleared.

   c) BELL (Ø7 hex): A sound should be made if possible when the CONSOLE: gets Ø7. If making a sound is not possible then ignore the Ø7.

d) SP (2Ø hex): Place a space at the current cursor position overwriting whatever is there. Move the cursor to the next column. If the cursor is on the last column of a line, it is best if the cursor stays where it is after the space fills that position. If the cursor is on the last column of the last line on the screen, it is also best if the cursor remains in that position and the screen does not scroll. These are the preferred actions of the cursor at end of line and end of screen; in the strict sense, the actions of the cursor in these circumstances are undefined.

e) NUL (ØØ hex): When a Null is sent to the CONSOLE: from the RSP, the CONSOLE: should delay for the ammount of time required to write one character but the state of the screen should not change.

f) All printable characters should be written to the screen and the cursor should move in the same way it does for SP.

g) See the discussion on pages 199-215 in the 1.1 Operating System Reference Manual for further requirements and information.

1Ø) CONSOLE: Input Requirements:

a) The RSP takes care of echoing characters to the screen typed from the CONSOLE: keyboard.

(below items optional: The Start/Stop, Flush & Break characters are redefinable; see 9g above for more information.)

b) The Start/Stop character is detected by CONCK and is used to stop all processing until the character is received a second time. When the character is received (see 9g above for more information) one should loop in CONCK continuing to process other characters until:

1. the S/S character is received again
2. the Break character is received

In case 1, the suspended processing should continue as it was before the first S/S was typed. Action needed for the Break character is described below. The S/S character is never returned to the RSP and CONSOLE: type-ahead, if implemented, should continue during the suspended state. Offset from SYSCOM to this character is 85 decimal. (This and the next 2 characters are redefinable by the Setup program and SYSCOM is the system area that keeps track of this information. The pointer to the start of SYSCOM is passed to the CONSOLE: init routine and is stored at F8..F9 hex.)

c) The Flush character will stop all output and echoing to the CONSOLE: until its second occurrence (see 9g above). CONCK detects this and must set a flag to tell the CONSOLE: output routine to ignore characters while the flag is set. If the CONSOLE: is re-initialized or a Break character is received, the flush state should be turned off. Flush is never returned to the RSP. Flush only stops CONSOLE: output, other processing continues. Offset from SYSCOM to this character is 83 decimal.

d) The Break character should cause CONCK to jump to the location stored at BF16. This location is also passed to the CONSOLE: init routine which stores it at BF16. The break character is never returned to the RSP and it should remove the system from Stop or Flush mode if it is in either mode. Offset from SYSCOM to this character is 84 decimal.

e) Type-ahead should be implemented in CONCK by storing characters typed at the keyboard in a queue until they are requested by a CONSOLE: read from Pascal. When the queue fills, further characters should be ignored and a bell sounded when they are typed. The length of the queue should be at least 2Ø characters.

11) For more information on CONSOLE: requirements, see pages 199-216 of the 1.1 Operating System Reference Manual.

b. Unit 6 (PRINTER:)

1) The interpreter takes care of expanding blank suppression codes (DLE's), EOF (the end of file character), and adding line feeds to every carriage return.

2) PRINTER: read, write and init have only the return address on the stack. PRINTER: status has the same items on the stack as CONSOLE: status. PRINTER: init should cause the PRINTER: to do a carriage return and a line feed and throw away any characters buffered to be printed. No form feed should be done.

3) For status, return in the first word of the status record the number of bytes buffered in the direction asked for; if this cannot be determined by your PRINTER:, return a Ø.

4) The PRINTER: write routine must buffer a line and send it all at once if your PRINTER: can only receive data that way.

5) Line Delimiter characters:

a) CR (hex ØD): A carriage return should cause the PRINTER: to print the current line and return the carriage to the first column. An automatic line feed should not be done by the PRINTER: driver when it reads a CR.

b) LF (hex ØA): The RSP will send line feeds to the PRINTER: driver after each carriage return. This should cause the PRINTER: to advance to the next line. If the PRINTER: must also do a carriage return when it is given a line feed, then this is O.K.

c) FF (hex ØC): This should cause the PRINTER: to move the paper to top of form and do a carriage return. If top of form is not possible on your PRINTER:, do a carriage return followed by a line feed.

6) It is assumed that input cannot be received from the PRINTER:. See the BIOS section for a discussion of how to get input from the PRINTER:. Normally, trying to get input from the PRINTER: should return completion error code #3.

c. Units 7 and 8 (REMOTE: in and REMOTE: out)

1) These must both go to the same driver.

2) The interpreter takes care of expanding blank suppression codes (DLE's), EOF and adding line feeds to every carriage return.

3) Same stack setup as the PRINTER:.

4) Status should return in first word of status vector the number of bytes buffered for the direction specified in the control word, return Ø if you have no way to check.

5) This unit is supposed to be an RS-232 serial line for many different applications so it is necessary that it transfer the data without modifying it in any way. The default transfer rate is 96ØØ baud.

6) It would be nice if the input to REMOTE: could be buffered in the same way input to the CONSOLE: is but this is not an absolute requirement.

7) REMOTE: init should set up the REMOTE: device so it is ready to read and write.

2. Block Structured Devices - Units 4 (boot unit), 5, 9, 1Ø, 11, 12.

   a. These units are assumed to be block structured devices, the drivers for these units must do their own Pascal Block to Track-Sector conversions.

      The UCSD system assumes the disk device is a Ø-based consecutive array of 512-byte logical blocks. All UCSD Pascal disks must have this logical structure no matter what their actual physical structure or size are. The physical allocation schemes for information on different types of disks are arranged with sectors that are of various sizes that depend on the hardware of the particular disk device used.

      The driver must convert the Pascal block number to the appropriate track and sector number of where that block is stored on its disk device. This could be a floppy or hard disk or some other type of device. It doesn't really matter, so long as your driver maps the Pascal Block to the correct place and continues to do so for the length (byte count) required for the UNITIO operation.

      The Pascal system uses logical blocks Ø and 1 for its bootstrap code. These logical blocks should not be used for anything else and should therefore only be available to Pascal through direct UNITREAD and UNITWRITE operations and not accessible by the system through any other means. This document will not attempt to describe the boot sequence and does not attempt to give you enough information to attach another driver or device to unit #4 so you can cold boot from that unit.

      When a UNITWRITE is done to disk where the byte count MOD 512 <> Ø (this means the last block included in the write would be partially written to according to the byte count), it is undefined whether garbage is written into the remaining part of this last block. So you may write a whole block anyhow if that is more efficient and the Pascal system will not suffer any bad consequences.

      When a UNITREAD is done from a disk you are not allowed to overwrite into the unused part of the last block (if there is an unused part due to byte count MOD 512 <> Ø). You must only send the number of bytes asked for because you could clobber memory having some other valid use if you wrote extra bytes. You will have to buffer the last sector inside your disk read routine then transfer exactly the number of bytes from this last sector needed to add up to the total bytes requested.

   b. The unit number will always be in the A-Register.

c. The stack setup for read and write is:

```
CONTROL WORD        (The MODE parameter mentioned in the Pascal
                     1.1 Language Reference Manual on page 41)
DRIVE NUMBER
BUFFER ADDRESS
BYTE COUNT
BLOCK NUMBER
RETURN ADDRESS  <--TOS
```

For init there is only the return address on the stack and for status the setup is the same as for the CONSOLE:.

d. Status requests should return the following in the status record:

word1:Number of bytes buffered in the direction asked for in the control word. Return Ø if you have no way of checking.
word2:Number of bytes per sector.
word3:Number of sectors per track.
word4:Number of tracks per disk.

3. Other - Unit 3

a. This unit has no meaning for the Apple II system except that UNITCLEAR on this unit sets text mode.

C. Considerations when attaching drivers for user defined devices numbers 128-143.

1. These unit numbers are provided for you to do whatever you want with them. you can define what they do except for the following protocols.

a. Follow the considerations for all drivers listed above.

b. The unit number will always be in the A-Register.

c. The stack setup for read and write is:

```
CONTROL WORD
DRIVE NUMBER
BUFFER ADDRESS
BYTE COUNT
BLOCK NUMBER
RETURN ADDRESS  <--TOS
```

2. For init there is only the return address on the stack and for status the setup is the same as for the CONSOLE:.

SAMPLE DRIVER FOR A USER DEFINED DEVICE

```
        ;Locations Ø..35 hex may be used as pure temps. One should never
        ;assume these locations won't be clobbered if you leave the envi-
        ;ronment of the driver itself. ("Leaving" includes calls to CONCK).

CONCKADR .EQU   Ø2

        ;Only one .PROC may occur in a driver, each driver ATTACHED must be
        ;assembled separately using the Pascal assembler and can have no
        ;external references.

        .PROC  U128DR
        STA    TEMP1        ;Save A-Reg contents (unit#)
        PLA
        STA    RETURN
        PLA
        STA    RETURN+1
        TXA                 ;Use the X reg to tell you what kind of call this is
        CMP    #2
        BEQ    INIT
        CMP    #4
        BEQ    STATUS
        CMP    #Ø
        BEQ    PMS
        CMP    #1
        BEQ    PMS

        ;Could have error code here
        JMP    RET

PMS     PLA                 ;Get the parameters
        STA    BLKNUM
        PLA
        STA    BLKNUM+1
        PLA
        STA    BYTECNT
        PLA
        STA    BYTECNT+1
        PLA
        STA    BUFADR
        PLA
        STA    BUFADR+1
        PLA
        STA    UNITNUM      ;Also in TEMP1
        PLA
        STA    UNITNUM+1    ;Should always be Ø
        PLA
        STA    CONTROL
        PLA
        STA    CONTROL+1
        TXA
        BNE    WRITE
```

```
READ     JSR    GOTOCK      ;Your driver's code for a read
                   (If more than one unit were attached to this driver, this
                    code could jump to various places depending on the contents
                    of the A-Reg stored in TEMP1)
         JMP    RET

WRITE    JSR    GOTOCK      ;Your driver's code for a write
         JMP    RET

         ;If you wanted to call CONCK whenever your device did a read
         ;or write, you would use this routine:

CKR      .WORD  CONCKRTN-1
GOTOCK   LDY    #55.        ;Offset to address of CONCK
         LDA    @ØE2,Y
         STA    CONCKADR
         INY
         LDA    @ØE2,Y
         STA    CONCKADR+1
         LDA    CKR+1       ;Set it up so you return to CONCKRTN after
         PHA                ;the CONCK call.
         LDA    CKR
         PHA
         JMP    @CONCKADR   ;Jump to CONCK
CONCKRTN RTS                ;Return to caller.

INIT     JMP    RET         ;Your driver's code for init

STATUS   PLA                ;Your driver's code for status
         STA     CONTROL
         PLA
         STA    CONTROL+1
         PLA
         STA    BUFADR      ;Address of status record.
         PLA
         STA    BUFADR+1

RET      LDA    RETURN+1
         PHA
         LDA    RETURN
         PHA
         LDA    TEMP1
         RTS

RETURN   .WORD  Ø           ;Can't use Ø page for these since we leave
TEMP1    .WORD  Ø           ;our environment when going to CONCK.
CONTROL  .WORD  Ø
UNITNUM  .WORD  Ø
BUFADR   .WORD  Ø
BYTECNT  .WORD  Ø
BLKNUM   .WORD  Ø

         .END
```

```
SAMPLE DRIVER FOR CONSOLE: DRIVER REPLACEMENT

ROUTINE  .EQU   Ø2
TEMP1    .EQU   Ø4

         .PROC  CKATCH
         JMP    CONCKHDL    ;SYSTEM.ATTACH will patch the start of CONCK to
                            ;jump here when you attach a driver to the CONSOLE:

                            ;We are not popping the return address from
                            ;the stack because we'll return from the system
                            ;routine we call from this driver.

         STA    TEMP1       ;All the read, write, init and stat calls will jump
                            ;here (starting address of your CONSOLE: driver+3)
         STY    TEMP1+1
         TXA
                            ;This example shows you how to have your own code
                            ;for the CONSOLE: as well as using the system
                            ;CONSOLE: routines.  If you want to replace the
                            ;system routines completely, you need to pull the
                            ;return address here.
         BEQ    READ
         CMP    #1
         BEQ    WRITE
         CMP    #2
         BEQ    INIT
         CMP    #4
         BEQ    STATUS

         ;Error code here

READ     LDY    #1          ;Your driver's code for a read - offset to address
                            ;of CONSOLE: read in the copy of the JMP vector made
                            ;by SYSTEM.ATTACH.  See the jump vectors in the BIOS
                            ;section below to see how we get the offsets.
         BNE    GET

         ;You would have a JMP RET here (see example for user defined device)
         ;if you were not using the system CONSOLE: routines as well.

WRITE    LDY    #4          ;Your driver's code for a write
         BNE    GET

INIT     LDY    #7          ;Your driver's code for init
         BNE    GET

STATUS   LDY    #43.        ;Your driver's code for status
```

```
GET       LDA    @ØE2,Y      ;At E2 is a pointer to the copy of the jump vector
                             ;made by SYSTEM.ATTACH before it was modified to
                             ;attach your drivers.
          STA    ROUTINE
          INY
          LDA    @ØE2,Y
          STA    ROUTINE+1
          LDY    TEMP1+1     ;Restore registers
          LDA    TEMP1
          JMP    @ROUTINE    ;Go to the original CONSOLE: driver for this
                             ;I/O command.  You will return from there-the
                             ;BIOS is already folded in due to the way your
                             ;driver was attached by SYSTEM.ATTACH.

CONCKHDL  PHP                ;Duplicate the 1st three instructions of CONCK
          PHA                ;as they were patched by SYSTEM.ATTACH to jump
          TXA below          ;to the 1st instruction of this driver.

          ;Here you can put the code for your own part of CONCK (you may want to
          ;check some additional device like a keypad or something or you may
          ;want to replace the system CONCK routine altogether.  If you do this,
          ;you must save the rest of the machine state and return it when you
          ;finished.  See example below.

          TYA                ;Save Y-Reg contents for a second.
          PHA

          ;This code gets us to the system CONCK routine.
          CLC
          LDY    #55.        ;Offset to the address of system CONCK in the
                             ;copy of the original JMP vector.
          LDA    @ØE2,Y
          ADC    #3          ;Add 3 so you enter right after the three
                             ;instructions you duplicated at CONCKHDL.
          STA    ROUTINE
          INY
          LDA    @ØE2,Y
          ADC    #Ø
          STA    ROUTINE+1
          PLA                ;Restore Y-Reg.
          TAY
          TXA                ;Last of CONCK instructions SYSTEM.ATTACH over-
                             ;wrote with the JMP to the start of this driver.
          JMP    @ROUTINE    ;Goto system CONCK and return from there.
          .END
```

Here is another alternative for the CONCKHDL part of the above program.

```
CKRTN     .WORD CONCKRTN-1

CONCKHDL  ;  1. If you don't care about type-ahead, this could be simply the
          ;     following code (assuming your CONSOLE: read gets a character
          ;     directly from your CONSOLE: device whenever it is called):
```

```
        PHP
        INC     RANDL       ;RANDL is a permanent word at BF13 used in
                            ;the built in random function.
        BNE     $1
        INC     RANDH       ;RANDH
$1      PLP
        RTS

        ;  2. If you want type-ahead, this code should check to see if there
        ;     is a character available and stuff it into a typeahead buffer.
        ;  3. If you are using this with the regular CONCK (extra keypad to
        ;     check or statistics for example), then you can do it this way.

        PHP                 ;Save state of machine
        PHA
        TXA
        PHA
        TYA
        PHA

        ;Put your driver's part of CONCK here (gives your driver priority)

        LDA     CKRTN+1     ;Set up things to return from reg CONCK
        PHA
        LDA     CKRTN
        PHA
        PHA                 ;Push garbage to account for other pushes done
        PHA                 ;in first three bytes of CONCK
        CLC                 ;Setup to call CONCK
        LDY     #55.        ;Offset to the address of system CONCK in the
                            ;copy of the original JMP vector.
        LDA     @ØE2,Y
        ADC     #3          ;Add 3 so you enter right after the three
                            ;instructions you duplicated at CONCKHDL.
        STA     ROUTINE
        INY
        LDA     @ØE2,Y
        ADC     #Ø
        STA     ROUTINE+1   ;In this example we don't have to worry about
                            ;the machine state here as we are restoring
                            ;it after we call CONCK
        JMP     @ROUTINE    ;Goto system CONCK and return to CONCKRTN

CONCKRTN PLA                ;Restore state of machine
        TAY
        PLA
        TAX
        PLA
        PLP
        RTS                 ;Return to the guy who called CONCK.
```

MAKING ATTACH.DRIVERS

1. Execute the standard 1.1 LIBRARY program.

2. The output code file should be ATTACH.DRIVERS or could be named something else and renamed ATTACH.DRIVERS when you put it on the boot disk.

3. For the Link code file use the code file of your first driver.

4. Copy its slot #1 into slot #Ø of ATTACH.DRIVERS.

5. As long as you have more drivers to add, use N(ew to get another Link code file and copy its slot #1 into slots 2, 3...15 of ATTACH.DRIVERS.

6. When done, type ´Q´ then ´N´ followed by a RETURN for the notice. See the 1.1 Operating System Reference Manual for further information on the LIBRARY program.

THE WORKINGS OF SYSTEM.ATTACH

If it is on the boot disk, SYSTEM.ATTACH is executed by the operating system (both regular 1.1 and runtime 1.1) before SYSTEM.STARTUP. The 1.1 runtime system will use a runtime version of SYSTEM.ATTACH.

The error messages that can be generated by SYSTEM.ATTACH are:

1. ERROR =>No records in ATTACH.DATA

2. ERROR =>Reading segment dictionary of ATTACH.DRIVERS

3. ERROR =>reading driver

4. ERROR =>A needed driver is not in ATTACH.DRIVERS

5. ERROR =>ATTACH.DATA needed by SYSTEM.ATTACH

6. ERROR =>ATTACH.DRIVERS needed by SYSTEM.ATTACH

If all goes well attaching drivers, SYSTEM.ATTACH will display nothing unusual in the regular boot sequence except for extra disk accesses and anything done in the init calls to any of the attached devices.

AN ADDITIONAL NOTE ON ATTACHING TO SYSTEM UNIT NUMBERS

There is a problem in attaching custom devices to the standard system unit numbers, 4-12. this problem will show up when you can communicate with the device via UNITREAD and UNITWRITE, but not with the system volume name. For example, UNITWRITE (6,....) will work, but WRITELN (F,....) where F has been opened to 'PRINTER:' will not work.

This problem is due to a table in the operating system which is set up before SYSTEM.ATTACH is called. To solve this, have the init routine in your attach drive stuff an 'I' in the type-ahead buffer only the very first time it is called. Then when asked if SYSTEM.ATTACH should call the init routine for this driver, reply 'Yes'. This will cause the routine that sets up the table to be executed after SYSTEM.ATTACH is executed.

II. BIOS

This section explains things in the BIOS area that are extensions and modifications that were added to Apple Pascal version 1.1 that were different or not there at all in Apple Pascal version 1.Ø (UCSD version II.1).

1. The disk routines have been modified to handle interrupts (So interrupt driven devices could be attached to 1.1 Pascal) if they are being used. To use interrupts, one would have to attach an interrupt driver, then patch the IRQ vector (FFFE hex) to point to this driver. The Pascal system is defined to come up with interrupts turned off, so once the driver is brought in and the IRQ patched, interrrupts must be turned on. The driver's init call could patch the IRQ and turn on interrupts. The disk routines save the current state of the system and turn interrupts off only during crucial time periods; the state of the system is returned during non-crucial time periods so interrupts can be handled. This has not been tested at this time, so there is no data concerning the maximum interrupt response time delay.

2. The control word parameter in UNITREAD and UNITWRITE was not passed on to the BIOS level routines from the RSP level. This has been done in 1.1 to allow the changes to the control word listed below under special character checking and also so user defined units or attached Pascal units can use the user defined bits of the control word.

3. IORESULTS 128-255 are available for user definition on user defined devices.

4. UNITSTATUS has been implemented in the Apple II Pascal 1.1 system. This works for the Pascal system units as described in the ATTACH part of this document. For user defined units, UNITSTATUS can be used for whatever is necessary.

   UNITSTATUS is a procedure that can be called from the Pascal level in the same way UNITREAD can. It has three parameters:

   a. unit number
   b. pointer to a buffer
      (any size buffer you want of type PACKED ARRAY OF CHAR)
   c. control word

   When you make a UNITSTATUS call from Pascal, the call should look like:

   ```
   UNITSTATUS (UNITNUM,PAC,CONTROL);
   ```

   Where UNITNUM and CONTROL are integers and PAC is a PACKED ARRAY OF CHAR or a STRING and may be subscripted to indicate a starting position for data transfer. See further information on what UNITSTATUS is defined to do for the various devices in the ATTACH part of this document.

The control word will tell the status procedure for a particular unit what information about the unit you want. Bit Ø of this word should =1 for input status and =Ø for output status. UNITSTATUS is implemented with bit 1 of the control word =1 meaning the call is for unit control. When this bit =Ø the call is for UNITSTATUS. In all cases, bits 2-12 are reserved for system use and bits 13-15 are available for user defined functions.

An entry in the jump vector has been made for each of the system UNITSTATUS calls, i.e. CONSOLESTAT, PRINTERSTAT, REMOTESTAT, etc. UNITSTATUS calls to a user defined device will all go through the same jump vector.

5. The handling of CTRL-C by the Apple BIOS was non-standard in 1.Ø. The UCSD BIOS definition specifies that a CTRL-C coming from REMOTE: or the PRINTER: should be placed in the input buffer and then no more characters should be received. Our BIOS did fill the buffer with nulls, including the place where the CTRL-C was to go. Apple Pascal's BIOS now conforms to the standard definition, where the null filling of the buffer is done only when CTRL-C comes from the CONSOLE: (#1:).

6. The UNITIO routines can be accessed from assembly procedures by pushing the correct parameters on the stack and using the jump vector to get to the BIOS routine. A separate document needs to be written describing how this is done and pointing out the problems doing it in the case of the CONSOLE:, SYSTERM:, PRINTER:, and REMOTE: units. These problems are concerned with the special character handling done in the RSP for these units. The assembly procedures calling the Pascal drivers for these units would either have to repeat portions of the RSP code themselves or not get the special character handling provided by the RSP. Calling the CONSOLE: init routine requires pointers to SYSCOM and the break routine to be passed on the stack. These pointers are now stored in a fixed location so assembly routines wanting to call CONSOLE: init can get at them. See the locations section.

7. Suppression of Special Character Checking. There are three types of special characters in the Pascal system:

   A. Characters used to control the 4Ø-column screen. These are CTRL-A, CTRL-Z, CTRL-W, CTRL-E, and CTRL- K.

   B. Pascal system control characters for general CONSOLE: use. These are CTRL-S and CTRL- F.

   C. Types A and B are checked for by the CONCK funtion in BIOS. There are other special characters checked for in the RSP. These are CTRL-C, DLE, and CR (line feeds are automatically appended to CR). With UNITREAD and UNITWRITE, the automatic handling done by the Pascal system of these characters can be turned off. To turn off DLE expansion and EOF checking, give bit 2 of the control word a value of 1. The automatic addition of line feeds to carriage returns can be suppressed by setting bit 3 of the control word to 1.

A way was needed to suppress special handling for types A nd B. This can now be done in two ways. First, the control word of UNITREAD/UNITWRITE will turn of checking for type A control characters if bit 4 is set and will turn off checking for type B characters if bit 5 is set. In this mode, the special character handling will only be turned off during that particular UNITIO. This will be be done for you in the RSP by setting bits in the byte 'SPCHAR' at location BF1C. The CONCK routine will look at bit Ø of SPCHAR and if set will not look for the type A control characters; if bit 1 is set, it will not look for the type B characters. If you set these bits in SPCHAR yourself instead of letting the RSP do it through the UNITIO control word, then the associated special character checking will be turned off until you reboot or reset the bits again. When special character checking is turned off, the characters are passed back to the Pascal level like all other characters would be. You can use these added features to redefine the system special characters in a particular application program or to just disable them.

8. The EOF character (CTRL-C) causes a lot of problems in the Pascal system. The cause of the problems is that the editor looks for this character to end many of its editing modes. The editor has its own GETCHAR routine which reads each character the user enters from SYSTERM:. When reading from SYSTERM: instead of the CONSOLE:, the EOF character is passed back as any other character but it still ends the current call to UNITREAD. The editor echoes each char to the CONSOLE: itself until it comes to CTRL-C. The operating system and the filer both use the GETCHAR routine in the operating system. This routine is defined to re-initialize the system if it gets a CTRL-C from the CONSOLE: and it reads from the CONSOLE:, not SYSTERM:. You must be sure not to end responses with CTRL-C except for the cases (in the editor only) that are supposed to end with CTRL-C. See the 1.1 Operating System Reference Manual.

9. The BIOS card-recognizing section has been enhanced to recognize a new FIRMWARE type card. This card will allow OEM's to have their drivers in their own firmware on the card. Routines have been added to allow for init, read, write, and status calls to this new type card. This protocol has been documented and is attached as an appendix to this document.

1Ø. As you can see, the Pascal system memory usage is scattered all over the 64K space. The Apple II was not designed with a stack machine, like the Pascal P-machine, in mind. We don't need any more constraints fixing certain pieces of the system to certain EXACT places. To make the best use of the space we have, we must have the ability to move things around. To achieve this goal, we intend the following:

    a. To stop people from writing things that peek here and poke there and expect things to stay exactly where they were for future versions.

b. Various people need space for patch areas and other purposes. All programs have to be written so this space does not have to be in a permanent fixed location if this is at all possible. The areas reserved for system use are filling up fast, we need to avoid using them. You can get space dynamically using NEW but you must be careful that this space stays around for the whole time you need it. If you are attaching a driver, you can get buffer space in the driver by using .WORD or .BLOCK in the Assembler. This space can be accessed from outside the driver if you know the offset to the start of this space from the start of the driver. This method could even be used to get space below the heap by attaching a driver to one of the user defined devices that is a large .BLOCK and is only used as a buffer. You can get the address of this buffer (of a driver) from the jump vector that has a pointer to the driver. Pointers to all the jump vectors are in zero page, see the locations section below.

c. The jump vector will have a fixed order for version 1.1 and future versions. The order is the same as in version 1.Ø with the new entries added to the bottom. The setup for the jump vector and getting into the BIOS is different than in 1.Ø. Here is how the new system is set up with the fixed order for the jump vector:

```
        ;  MAIN BIOS JUMP TABLE CALLED FROM INTERPRETER
        ;  (FOLLOWED BY REAL JUMP TABLE AT FIXED OFFSET)
        ;  RSP CALLS COME TO THIS JUMP VECTOR

BIOS    JSR    SAVERET    ;CONSOLE READ  (Jump vector before fold)
        JSR    SAVERET    ;CONSOLE WRITE
        JSR    SAVERET    ;CONSOLE INIT
        JSR    SAVERET    ;PRINTER WRITE
        JSR    SAVERET    ;PRINTER INIT
        JSR    SAVERET    ;DISK WRITE
        JSR    SAVERET    ;DISK READ
        JSR    SAVERET    ;DISK INIT
        JSR    SAVERET    ;REMOTE READ
        JSR    SAVERET    ;REMOTE WRITE
        JSR    SAVERET    ;REMOTE INIT
        JSR    SAVERET    ;GRAFIC WRITE
        JSR    SAVERET    ;GRAFIC INIT
        JSR    SAVERET    ;PRINTER READ
        JSR    SAVERET    ;CONSOLE STAT
        JSR    SAVERET    ;PRINTER STAT
        JSR    SAVERET    ;DISK STAT
        JSR    SAVERET    ;REMOTE STAT
KCONCK  JSR    SAVERET    ;To get to CONCK from CONCKVEC
        JSR    SAVERET    ;USER READ  (For UDRWIS)
                          ;USER WRITE
                          ;USER INIT
                          ;USER STAT
        JSR    SAVERET    ;For PSUBDR
        JSR    SAVERET    ;IDSEARCH
               etc.
```

```
        ;  THIS JUMP TABLE MUST BE OFFSET FROM BIOSTBL BY EXACTLY $5C.
        ;  SYSTEM.ATTACH MODIFIES THIS JUMP VECTOR TO POINT TO ATTACHED DRIVERS
        ;  FOR THE STANDARD SYSTEM UNITS.

BIOSAF  JMP     CREAD       ;Jump vector after fold.
        JMP     CWRITE
        JMP     CINIT
        JMP     PWRITE
        JMP     PINIT
        JMP     DWRITE
        JMP     DREAD
        JMP     DINIT
        JMP     RREAD
        JMP     RWRITE
        JMP     RINIT
        JMP     IORTS       ;Do nothing for GRAFWRITE.
        JMP     GRAFINIT
        JMP     IORTS       ;Do nothing for PRINTER: read.
        JMP     CSTAT
        JMP     ZEROSTAT    ;For PRINTER: stat, pop parameters and store Ø
                            ;in 1st buffer word.
        JMP     DSTATT
        JMP     ZEROSTAT    ;For REMOTE: stat, pop parameters and store Ø
                            ;in 1st buffer word.
        JMP     CONCK
        JMP     UDRWIS      ;Routine to get to user defined devices, see ATTACH
                            ;part of document for description of this routine
        JMP     PSUBDR      ;Routine to get to drivers that are substituted
                            ;for the standard Pascal disk units 4, 5, 9..12.
                            ;See ATTACH part of document for description of
                            ;this routine.
        JMP     IDS

        ; STRIP LOCAL RETURN ADDRESS, STRIP PASCAL ADDRESS AND
        ; SAVE IN RETL, RETH.  PLACE ´GOBACK´ ON RETURN STACK THEN RESTORE
        ; LOCAL RETURN ADDRESS AND RETURN.  MEANWHILE UNFOLD BIOS INTO DXXX

SAVERET STA     TT1         ;SAVE A REGISTER
        PLA
        CLC
        ADC     #Ø5A        ;ADD OFFSET TO JUMP TABLE (BIOSAF)
        STA     TT2         ;LOCAL RETURN ADDRESS
        PLA
        ADC     #Ø
        STA     TT3
        PLA
        STA     RETL        ;PRESERVE PASCAL RETURN
        PLA
        STA     RETH
        .IF     RUNTIME=Ø
        LDA     ØCØ83       ;UNFOLD BIOS INTO DXXX
        .ENDC
        LDA     TT1         ;RESTORE A-REG
        JSR     SAVRET2     ;PUTS ´GOBACK´ ON STACK
```

```
        ; FOLD INTERP INTO DXXX THEN RETURN TO PASCAL VIA RETURN
        ; ADDRESS SAVED IN RETL, RETH

GOBACK  STA     TT1         ;SAVE A-REG
        LDA     RETH
        PHA
        LDA     RETL
        PHA
        .IF     RUNTIME=Ø
        LDA     ØCØ8B       ;FOLD INTERP INTO DXXX
        .ENDC
        LDA     TT1
        RTS                 ;AND BACK TO PASCAL
SAVRET2 JMP     @TT2        ;JUMP INTO JUMP TABLE (BIOSAF)
```

d. In zero page are two words pointing to the base of the two jump vectors (before and after the fold). These are stored in PERMANENT locations that had a value of Ø in the old 1.Ø release and were not used by the system (see locations section). Applications needing to patch the jump vectors can store the offset from the vector base in the Y-Register and use indirect indexed addressing to do the patch. The application will need to have the vector base locations for the old release hard-coded in as the base pointer for the old 1.Ø release will be Ø. If you want to write an application that works with 1.Ø and 1.1 and future versions, you know if the zero page vector pointers are Ø, it's the 1.Ø system, otherwise it's 1.1 or a future version which will use the same protocols as 1.1 as described in this document.

   It is important that any application patching the jump vector temporarily then returning it to its original value get the original value from the vector itself before the patch and put it in a storage location. When the vector needs to be restored to its original state, use this storage location for its original value. The patches should be done in this manner so the applications doing the patches will always return the system to its original state no matter what past, present or future Pascal version it is patching.

e. For CONSOLE: init to be used from assembly routines the locations of SYSCOM and the BREAK routine have to be available. The CONINIT routine requires these on the stack. Pointers to SYSCOM and BREAK will be stored by the interpreter boot in a PERMANENT location in the BFØØ page (see locations section).

f. Since the old 1.Ø release, the code to jump to the CONCK routine has been set up at location BFØA. Anyone wishing to get to the CONCK routine should do a JSR BFØA as this will always get them there no matter where the CONCK routine really is. The keypress function has been changed to conform to this new convention, but it will use the old convention if it is working from within an old system. Do not try to get to CONCK in this way from within an ATTACHED driver as you will lose your return address to Pascal. See ATTACH part of this document for how to get to CONCK from an attached driver.

g. There is now a version byte so one can tell which version (1.Ø, 1.1, etc.) of Apple Pascal he is working with. There is also a "flavor" byte to tell one which flavor of this version he has (regular, runtime, runtime without sets, etc.) (see locations section).

11. Whenever SYSTEM.ATTACH is used, it will make a copy of the original BIOS jump vector (the after fold vector that has the actual driver addresses in it) and put this below the heap with the drivers that are attached. It will leave a pointer to this copy of the vector at location ØØE2. You can use this vector in your drivers to get to the standard Apple drivers for any device. This way you can define a driver that does something above and beyond the standard Apple driver, yet this new driver can still make use of the standard Apple driver. See the ATTACH part of this document for more information.

12. In the RSP are two vectors that tell the RSP what is legal (input and/or output) for a particular character oriented device (CONSOLE:, REMOTE:, and PRINTER:). For example it tells the RSP that it is illegal to read from the PRINTER:. If you wanted to ATTACH a PRINTER: driver so you could read from the PRINTER:, you would have to change this vector. ØØE4 points to the READTBL vector and ØØE6 to the WRITTBL vector. Let's take the READTBL for an example:

```
READTBL  ;table of routine addresses to be called when writing to that unit
         ;(disk I/O does not use this table).
         ;an entry=Ø means that the operation is illegal for that unit.

         .WORD  BIOS+CONREAD        ;unit 1
         .WORD  BIOS+CONREAD        ;unit 2
         .WORD  Ø                   ;unit 3
         .WORD  Ø                   ;units 4 and 5 are disk units
         .WORD  Ø
         .WORD  Ø                   ;unit 6 is PRINTER:
         .WORD  BIOS+REMREAD        ;unit 7
         .WORD  Ø                   ;unit 8 is REMOTE write which has
                                    ;an address in the WRITTBL
```

Here BIOS refers to the base of the jump vector before the fold and CONREAD is the offset off the base of that vector to get to the jump to the CONSOLE: read routine (for CONSOLE: read the offset is Ø, for CONSOLE: write it's 3, etc). The value for BIOS is the pointer stored in location ØØEC mentioned in the locations section below.

III. LOCATIONS

These are the locations of new system permanents mentioned in this document, all pointers are set up by the system and are stored low byte first. Do not modify what is stored in these pointers (except for SPCHAR if you want to suppress special character checking) since the system uses this information too. These locations are defined to have the same function & remain in the same place for future versions of Apple II Pascal.

BF1C	SPCHAR	(To control special characters)
BF1D	IBREAK	(Set by boot in interp for assembly calls to CONINIT)
BF1F	ISYSCOM	(Set by boot in interp for assembly calls to CONINIT)
BF21	VERSION	(1 byte Version number of system, =2 for the new release, =Ø for version 1.Ø)
BF22	FLAVOR	(This byte tells which flavor [runtime, regular, etc.] of this VERSION you are dealing with) The encoding is: 1 -->regular system runtime versions: 2 -->LC-ALL (LC- means no language card) 3 -->LC-no sets 4 -->LC-no floating point 5 -->LC-no sets or floating point 6 -->LC+ALL 7 -->LC+no sets 8 -->LC+no floating point 9 -->LC+no sets or floating point Ø -->the old 1.Ø release.
BFCØ-BFFF	BDEVBUF	(Area for non-Apple boot devices, like the CORVUS)
ØØE2	ACJVAFLD	(Pointer to ATTACH copy of the original Jump Vector after the fold)
ØØE4	RTPTR	(Pointer to READTBL)
ØØE6	WTPTR	(Pointer to WRITTBL)
ØØE8	UDJVP	(Pointer to user device jump vector)
ØØEA	DISKNUMP	(Pointer to disknum vector)
ØØEC	JVBFOLD	(Pointer to jump vector before fold)
ØØEE	JVAFOLD	(Pointer to jump vector after fold)

FFF6	(Version word which = 1 for version 1.Ø = Ø for version 1.1 This version word should not be used at runtime to tell which version you have. For that use the version byte mentioned above. This word should only be used by software that wants to see which SYSTEM.APPLE it is dealing with by looking at the contents of this word in the SYSTEM.APPLE file when it is not loaded in memory)
FFF8	(Start vector)
FFFA	(NMI non maskable interrupt vector)
FFFC	(RESET vector)
FFFE	(IRQ interrupt request vector)

The locations and code in the 1.Ø ´PRELIMINARY APPLE PASCAL GUIDE TO INTERFACING FOREIGN HARDWARE´ BIOS document are NOT the same for Apple Pascal 1.1 and that document clearly stated we would not commit ourselves to keeping them the same.

IV. PASCAL 1.1 FIRMWARE CARD PROTOCOL

One major problem with Apple Pascal 1.Ø is the way it handles peripheral cards. It was set up to work with the four peripheral cards that Apple supported at the time of its release (the disk, communciations, serial and parallel cards) and had no mechanism for interfacing any other devices. Since Apple, as well as many other vendors, continue to produce new peripherals for the Apple II, a new protocol was designed and implemented in the Pascal 1.1 BIOS which allows new peripheral cards to be introduced to the system in a consistent and transparent fashion. The new protocol is called the "firmware card" protocol, since the BIOS deals with these cards by making calls to their firmware at entry points defined by a branch table on the card itself. The new protocol fully supports the Pascal typeahead function and KEYPRESS will work with firmware cards used as CONSOLE devices. The following paragraphs describe the firmware card protocol in full detail.

A firmware card may be uniquely identified by a four byte sequence in the card's $CNØØ ROM space. Location $CNØ5 must contain the value $38 and location $CNØ7 must contain $18. Note that these are identical to the Apple Serial Card. A firmware card is distinguished from a serial card by the further requirement that location $CNØB must contain the value $Ø1. This value is called the "generic signature" since it is common to all firmware cards. The value at the next sequential location, $CNØC, is called the "device signature", since it uniquely identifies the device.

The device signature byte is encoded in a meaningful way. The high order four bits specify the class of the device while the low order four bits contain a unique number to distinguish between specific devices of the same class. The appendix to this document defines some device class numbers; in any case, vendors should contact Apple Technical Support to make sure they use a unique number for their device signature. Although the device signature is ignored by the 1.1 BIOS, it may be used by applications programs to identify specific devices.

Following the two signature bytes is a list of four entry point offsets, starting at address $CNØD. These four entry points must be supported by all firmware cards. They are the initialization, read, write and status calls. The BIOS takes care of disabling the $C8ØØ ROM space of all other cards before calling the firmware routines.

The offset to the initialization routine is at location $CNØD. Thus, if $CNØD contains XX, the BIOS will call $CNXX to initialize the card. On entry, the X-Register contains $CN (where N is the slot number) and the Y-Register contains $NØ. On exit, the X-Register should contain an error code, which should be Ø if there was no error. This error code is passed on to the higher levels of the system in the global variable IORESULT. Registers do not have to be preserved.

The offset to the read routine is at location $CNØE. On entry, the X-Register will contain $CN and the Y-Register will contain $NØ. On exit, the A-Register should contain the character that was read while the X-Register contains the IORESULT error code. The A- and Y-Registers do not have to be preserved.

The offset to the write routine is at location $CNØF. On entry, the A-Register contains the character to be written while the X-Register contains $CN, and the Y-Register contains $NØ. On exit the X-Register should contain the IORESULT error code (which should be Ø for no error). The A- and Y-Registers do not have to be preserved.

The offset to the status routine is at location $CN1Ø. On entry, the X-Register contains $CN and the Y-Register contains $NØ while the A-Register contains a request code. If the A-Register contains Ø, the request is "Are you ready to accept output?". If the A-Register contains 1, the request is "Do you have input ready for me?". On exit, the driver returns the IORESULT error code in the X-Register and the results of the status request in the carry bit. The carry clear means FALSE (i.e., no, I don't have any input for you), while the carry set means TRUE. Note that the status call must preserve the Y-Register, but does not have to preserve the A-Register.

Thus, sample code for the first few bytes of a firmware card's $CNØØ space should look something like:

```
BASICINIT     BIT   ØFF58         ;set the v-flag
              BVS   BASICENTRY    ;always taken
IENTRY        SEC                 ;BASIC input entry point
              DFB   $9Ø           ;opcode for BCC
OENTRY        CLC                 ;BASIC output entry point
              CLV
              BVC   BASICENTRY    ;Always taken

              ;  Here is the Pascal 1.1 Firmware Card Protocol Table

              DFB   $Ø1           ;Generic signature byte
              DFB   $41           ;Device signature bye

PASCALINIT    DFB   >PINIT        ; > means low order byte
PASCALREAD    DFB   >PREAD        ;offset to read
PASCALWRITE   DFB   >PWRITE       ;offset to write
PASCALSTATUS  DFB   >PSTATUS      ;offset to status routine
```

The above code fulfils all the requirements for both the BASIC and Pascal 1.1 I/O protocols. The routines PINIT, PREAD, etc, are probably jumps into the card's $C8ØØ space which is already properly enabled by the BIOS. The reason the $CNØØ space was chosen for the protocol (as opposed to the $C8ØØ space) is that the BASIC protocol requires that all cards have $CNØØ ROM space while some smaller cards may not need any $C8ØØ ROM space.

The firmware card protocol includes two optional calls that do not have to be implemented but would be nice. The BIOS checks location $CN11 to determine if the optional calls are present; if that location contains a $ØØ, then the BIOS thinks the calls are implemented. Thus if your card does not implement the optional calls, you should ensure that $CN11 contains a non-zero value. The two optional calls are a control call pointed to by $CN12 and an interrupt handler call pointed to by $CN13.

The control call entry point is specified by the offset at $CN12. On entry, the X-Register contains $CN, the Y-Register contains $NØ and the A-Register contains the control request code. Control requests are defined by the device. On exit, the X-Register should contain the IORESULT error code.

The interrupt poll entry point is specified by the offset at $CN13. On entry, the X-Register contains $CN and the Y-Register contains $NØ. The interrupt poll routine should poll the card's hardware to determine if it has a pending interrupt; if it does not it should return with the carry clear. If it does, it should handle the interrupt (including disabling it) and return with the carry set. Also, the X-Register should contain the IORESULT error code, which should be Ø if there was no error. An interrupt polling routine must be careful not to clobber any zero page or screen space temporaries.

The control and interrupt requests are not implemented in the Pascal 1.1 BIOS, but it would be nice to support them if possible as they may be implemented in later versions of the Pascal BIOS as well as other forthcoming operating system environments for the Apple II.

Note that the firmware card signature is a superset of the Apple serial card signature as recognized by the Pascal 1.Ø BIOS. This allows a firmware card to function with both Pascal 1.Ø and Pascal 1.1. If a card wishes to work with Pascal 1.Ø as a "fake" serial card, it must provide an input entry point at $C84D and an output entry point at $C9AA. Note that since Pascal 1.Ø will think the card is a serial card, type-ahead and KEYPRESS capabilities will be lost.

ADDITIONAL NOTES

1. The Pascal RSP expects the high order bit of every ASCII character it receives from the CONSOLE: read routine to be clear. The RSP will not do this for you; you must ensure the high bit of all text your card passes to the RSP from the CONSOLE: read routine is clear.

2. Zero page locations $ØØ to $35 may be used as temporaries by your firmware, as are the slot Ø screen space locations ($478, $4F8, etc.). In general, peripheral card firmware should be as conservative as possible in their memory usage, preserving zero page contents whenever possible. An interrupt polling routine must not destroy these or any other memory locations.

3. Location $7F8 must be set up to contain the value $CN, where N is the slot number, if your card utilizes the $C8ØØ expansion ROM space. The BIOS does not do this for you; his must be done if you want your card to function in an interrupting environment.

4. The firmware card status routine should be as quick as possible, as it may be called from within the I/O polling loops of many other peripherals if your card is being used as the console device. In no case should the status routine take longer than 1ØØ milliseconds.

5. A firmware card in slot 1 is automatically recognized as the volume "PRINTER:". A firmware card in slot 2 is automatically recognized as the volumes "REMIN:" and "REMOUT:". A firmware card in slot 3 is automatically recognized as the volumes "CONSOLE:" and "SYSTERM:".

APPENDIX

The following numbers correspond to device classes used in the device signature code. Make sure you contact Apple Technical Support to ensure that you have a unique device signature code.

Ø -- reserved
1 -- printer
2 -- joystick or other X-Y input device
3 -- I/O serial or parallel card
4 -- modem
5 -- sound or speech device
6 -- clock
7 -- mass storage device
8 -- 8Ø column card
9 -- Network or bus interface
1Ø -- Special purpose (none of the above)

11 through 15 are reserved for future expansion

ADDITIONAL INFORMATION

1. The type ahead buffer is located at $Ø3B1 hex and is $4E hex in length. It is implemented with a read pointer (RPTR at BF18 hex) and a write pointer (WPTR at $BF19 hex). At CONSOLE: init time, these should both be set to Ø. When a character is detected by CONCK, the WPTR is incremented then compared with $4E. If it is equal to $4E, it is set to $Ø (this is a circular buffer). Then the WPTR is compared with RPTR and if they are equal the buffer is full. If the buffer is not full, the character is stored at $Ø3B1+the value in WPTR.

   When removing a character from the type ahead buffer, use the following sequence. Compare the RPTR with WPTR and if they are equal, the buffer is empty and you must wait until a character is available from the keyboard. If they are not equal, increment the RPTR and compare it to $4E. If it equals $4E, set it to $Ø. Now get the character from location $Ø3B1+the value in RPTR.

   If you are implementing your own type-ahead, you can do it however you wish. This information is made available in case you want to check for input from another device as well as the standard system CONSOLE: and have characters from that device be put in the system type ahead buffer.

2. The example drivers in this document did not show the setting of IORESULT in the X-Register. This would be done in the code specific to your driver and should always be set to something (Ø if there are no errors). If there are errors, set it as described elsewhere in this document and the Pascal Manuals.

3. For further information, see the newest edition of the Apple II Reference Manual.

4. These listings from the BIOS are included to show you how we implemented certain system drivers. You cannot rely on the locations of these to stay in the same place in the BIOS in future releases of Apple II Pascal, nor can you rely on the routines themselves staying the same. They are included only as examples and to give you information that may not be documented elsewhere. This is not a complete BIOS listing so you may find references to routines or locations that are not included in this listing. The only locations that will be sure to remain the same for future releases are those mentioned in the LOCATIONS section above. We are against you poking the BIOS yourself to change or overwrite any of these routines. We did not include this information so you could poke the BIOS. If you do modify BIOS, it is completely at your own risk! We have provided the ATTACH utility so you can add your own drivers to the system without poking the BIOS and this is the way it should be done! If you have special requirements that are not solved by ATTACH, please contact Apple Technical Support, in writing.

```
            ;---------------------------------------
            ;
            ; ZERO PAGE PERMANENTS
            ;
            ;---------------------------------------

FIRST       .EQU  ØFØ          ;START ZERO PAGE USE
BAS1L       .EQU  FIRST        ;SCREEN 1 PTR
BAS1H       .EQU  FIRST+1
BAS2L       .EQU  FIRST+2      ;SCREEN 2 PTR
BAS2H       .EQU  FIRST+3
CH          .EQU  FIRST+4      ;HORIZ CURSOR, Ø..79
CV          .EQU  FIRST+5      ;VERT CURSOR, Ø..23
TEMP1       .EQU  FIRST+6
TEMP2       .EQU  FIRST+7
SYSCOM      .EQU  FIRST+8      ;2 BYTES PTR TO SYSCOM AREA

            ;---------------------------------------
            ;
            ; BFØØ PAGE PERMANENTS
            ;
            ;---------------------------------------

CONCKVECTOR .EQU  ØBFØA        ;4 BYTES
SCRMODE     .EQU  ØBFØE
LFFLAG      .EQU  ØBFØF
NLEFT       .EQU  ØBF11
ESCNT       .EQU  ØBF12
RANDL       .EQU  ØBF13
RANDH       .EQU  ØBF14
CONFLGS     .EQU  ØBF15
BREAK       .EQU  ØBF16        ;2 BYTES
RPTR        .EQU  ØBF18        ;1 BYTE
WPTR        .EQU  ØBF19        ;1 BYTE
RETL        .EQU  ØBF1A
RETH        .EQU  ØBF1B
SPCHAR      .EQU  ØBF1C        ;ØØ MEANS DO ALL SPECIAL CHARACTER CHECKING
                               ;Ø1 MEANS DON'T CHECK FOR APPLE SCREEN STUFF
                               ;Ø2 MEANS DON'T CHECK FOR OTHER SCREEN STUFF
IBREAK      .EQU  ØBF1D        ;INTERP STORES BREAK AND SYSCOM ADDRESS HERE FOR
ISYSCOM     .EQU  ØBF1F        ;USER ROUTINES TO GET AT
VERSION     .EQU  ØBF21        ;VERSION OF SYSTEM SET TO 2 FOR APPLE 1.1
FLAVOR      .EQU  ØBF22        ;SEE TABLE IN INTERP BOOT
SLTTYPS     .EQU  ØBF27        ;BF27..ØBF2E
XITLOC      .EQU  ØBF2F        ;INTERP INITS THIS TO LOCATION OF XIT
                               ;FORTRAN PROTECTION USES BF56..BF7F
                               ;VENDOR BOOT DEVICES CAN USE BFCØ..BFFF
```

```
            ;---------------------------------------
            ;
            ; MISCELLANEOUS PROGRAM EQUATES
            ;
            ;---------------------------------------

BUFFER      .EQU  Ø2ØØ          ;TEMP HSHIFT BUFFER (OVERLAPS DISK BUFFER)
CONBUF      .EQU  Ø3B1          ;78 CHAR TYPE-AHEAD BUFFER
CBUFLEN     .EQU  Ø4E           ;78 DECIMAL
NCTRLS      .EQU  14.           ;NUMBER OF CTRL CHARACTERS IN TABLE
SIGVALUE    .EQU  1
BYTEPSEC    .EQU  256.          ;DISK INFO FOR DISKSTAT
SECPTRAK    .EQU  16.
TRAKPDSK    .EQU  35.
UDJVP       .EQU  ØE8           ;Ø PAGE JUMP VECTOR POINTER LOCATIONS
DISKNUMP    .EQU  ØEA
JVBFOLD     .EQU  ØEC
JVAFOLD     .EQU  ØEE
HCMODE      .EQU  ØE1           ;THESE TWO BYTES USED FOR HIRES STUFF
HSMODE      .EQU  ØEØ

JVECTRS     .WORD UDJMPVEC
            .WORD DISKNUM
            .WORD BIOS
            .WORD BIOSAF

            ;---------------------------------------
            ;
            ; HARD RESET INITIALIZATION
            ;
            ;---------------------------------------

START       CLD                 ;SET HEX MODE
            SEI                 ;MAKE SURE INTERRUPTS ARE OFF.

            ;---------------------------------------
            ;
            ; CLEAR ALL MEMORY Ø TO BFFF
            ; (RUN-TIME SYSTEM:Ø TO TOPMEM + BF PAGE);
            ;
            ;---------------------------------------

            LDA   #Ø
            STA   ZEROL
            STA   ZEROH
            TAY
            TAX
```

```
ZERLP       STA   (ZEROL),Y   ;WRITE A BYTE OF Ø
            INY               ;BUMP POINTER
            BNE   ZERLP       ;LOOP TILL NEXT PAGE
            INC   ZEROH       ;BUMP MSB POINTER
            INX
            .IF   RUNTIME=1
            CPX   #TOPMEM     ;DONE CLEARING MEMORY?
            BNE   $1
            LDX   #ØBF        ;CLEAR BF PAGE
            STX   ZEROH
$1:         CPX   #ØCØ
            BNE   ZERLP
            .ELSE
            CPX   #ØCØ        ;DONE CLEARING BFXX?
            BNE   ZERLP
            .ENDC

            ;--------------------------------------
            ;
            ; CHECKSUM PROMS ON EACH SLOT TO FIND OUT WHO'S OUT THERE
            ; SUM TWICE TO TELL IF CARD THERE IF SUMS DON'T MATCH
            ; THEN NO PROM IS THERE IF MS BYTE OF SUM=Ø THEN NO PROM IS PRESENT
            ;
            ;--------------------------------------

            LDY   #ØC7        ;POINT TO SLOT 7 PROM
NXTCRD      STY   CKPTRH      ;(CKPTRL=Ø FROM MEM CLEAR)
            JSR   CKPAGE      ;16 BIT SUM IN X,A
            STA   CHECKL
            STX   CHECKH      ;SAVE FOR MATCH
            JSR   CKPAGE      ;SUM AGAIN
            CPX   #Ø          ;WAS MSB ZERO?
            BEQ   NOPROM      ;YES NO PROM ON CARD
            CMP   CHECKL      ;LSB MATCH?
            BNE   NOPROM      ;NO, NO PROM ON CARD
            CPX   CHECKH
            BNE   NOPROM      ;MSB DIDNT MATCH
            BEQ   SKIPIORTS   ;ALWAYS TAKEN

            ;--------------------------------------
            ;
            ; TABLE OF CNØ5 AND CNØ7 BYTES OF EACH CARD
            ;
            ;--------------------------------------

CNØ5BYTS    .BYTE ØØ3,Ø18,Ø38,Ø48
CNØ7BYTS    .BYTE Ø3C,Ø38,Ø18,Ø48
```

```
            ;---------------------------------------
            ;
            ; NOW THAT WE KNOW A CARD IS THERE, EXAMINE CNØ5 AND CNØ7 BYTE TO
            ; DETERMINE WHICH CARD IT IS.  SET CARDTYPE AS FOLLOWS:
            ;   Ø=CKSUM NOT REPEATABLE OR MSB=Ø
            ;   1=CKSUM REPEATABLE,CARD NOT RECOGNIZED
            ;   2=DISK CARD (BYTE Ø7= Ø3C)
            ;   3=COM CARD  (BYTE Ø7= Ø38)
            ;   4=SERIAL    (BYTE Ø7= Ø18)
            ;   5=PRINTER   (BYTE Ø7= Ø48)
            ;   6=FIRMWARE  (BYTE Ø7= Ø48)
            ;
            ;---------------------------------------
SKIPIORTS   LDX   #5          ;4 TYPES OF CARDS
NXTYP       LDY   #5          ;CHECK BYTE CNØ5 OF CARD
            LDA   (CKPTRL),Y
            CMP   CNØ5BYTS-2,X ;MATCH TABLE?
            BNE   TRYNXT      ;NO, TRY NEXT IN LIST
            LDY   #7
            LDA   (CKPTRL),Y  ;TEST CNØ7 BYTE
            CMP   CNØ7BYTS-2,X ;MATCH TABLE?
            BEQ   STOR        ;BOTH MATCHED, CARD RECOGNIZED
TRYNXT      DEX               ;BUMP TO NEXT IN LIST
            CPX   #2          ;TRY ALL TYPES IN LIST
            BCS   NXTYP       ;IF NOT IN LIST,FALL THRU WITH X=1
STOR        CPX   #4          ;IS IT A SERIAL CARD?
            BNE   STOR1
            LDY   #ØB
            LDA   (CKPTRL),Y
            CMP   #SIGVALUE
            BNE   STOR1
            LDX   #6
STOR1       LDY   CKPTRH
            TXA
            STA   SLTTYPS-ØCØ,Y
NOPROM      LDY   CKPTRH
            DEY               ;BUMP TO NEXT LOWER SLOT
            CPY   #ØCØ        ;SLOTS 7 DOWNTO 1 DONE?
            BNE   NXTCRD      ;LOOP TILL 7 SLOTS DONE, LEAVE WITH A-Reg:=Ø

            ;---------------------------------------
            ;
            ; SET UP CONCK VECTOR FOR KEYPRESS FUNCTION
            ;
            ;---------------------------------------

            BEQ   $2          ;ALWAYS BRANCHES
$1          JSR   KCONCK      ;HERE ARE THE 2 INSTRUCTIONS TO BE TRANSFERRED
            RTS
$2          LDY   #3          ;TRANSFER 4 BYTES TO BFØA
$21         LDA   $1,Y
            STA   CONCKVECTOR,Y
            DEY
            BPL   $21
```

```
            ;---------------------------------------
            ;
            ;SET UP JUMP VECTOR POINTERS IN Ø PAGE
            ;
            ;---------------------------------------

            LDY   #7
$3          LDA   JVECTRS,Y
            STA   UDJVP,Y
            DEY
            BPL   $3

            ;---------------------------------------
            ;
            ; SET SCREEN MODE ETC
            ;
            ;---------------------------------------

            LDA   #8Ø
            STA   HCMODE
            LDA   ØC Ø51        ;SET TEXT MODE
            LDA   ØC Ø52        ;SET BOTTOM 4 GRAFIX
            LDA   ØC Ø54        ;SELECT PRIMARY PAGE
            LDA   ØC Ø57        ;SELECT HIRES GRAFIX
            LDA   ØC Ø1Ø        ;CLEAR KEYBOARD STROBE
            JSR   FORM          ;ERASE SCREEN
            JSR   INVERT        ;PUT CURSOR ON SCREEN
            JSR   DRESET        ;DO ONCE ONLY DISK INIT
            LDA   SLTTYPS+3     ;WHAT CARD IN SLOT 3?
            LDY   #Ø3Ø          ;SLOT 3
            JSR   GENIT         ;SET BAUD IF COM OR SERIAL CARD THERE
            CPX   #Ø            ;WAS AN EXTERNAL CONSOLE THERE?
            BNE   STARTUP       ;NO, USE APPLE SCREEN
            LDA   #4
            STA   SCRMODE       ;SET BIT 2 FOR EXTERNAL CONSOLE
STARTUP     JMP   JPASCAL       ;FOLD IN INTERP AND START PASCAL

            ;---------------------------------------
            ;
            ; SUB TO CHECKSUM ONE PAGE
            ;
            ;---------------------------------------

CKPAGE      LDA   #Ø
            TAX                 ;CLEAR SUM
            TAY                 ;CLEAR INDEX
CKNX        CLC
            ADC   (CKPTRL),Y    ;ADD BYTE
            BCC   NOCRY
            INX                 ;INC HI BYTE IF CARRY
NOCRY       INY                 ;BUMP INDEX
            BNE   CKNX          ;SUM 256 BYTES
            RTS                 ;RETURN SUM IN X,A AND Y=Ø
```

```
                ;---------------------------------------
                ;
                ; BIOS HANDLERS FOR LOGICAL AND PHYSICAL DEVICES
                ;
                ;
                ; CONSOLE CHECK FOR CHARACTER AVAILABLE
                ; STATUS AND ALL REGISTERS PRESERVED
                ; IF CHAR AVAIL, PUT IN CONBUF AND INC WPTR.
                ;
                ; WARNING...THIS ROUTINE ALSO CALLED FROM DISK ROUTINES
                ;
                ;---------------------------------------

CONCK           PHP
                PHA
                TXA
                PHA
                TYA
                PHA
RNDINC          INC   RANDL         ;BUMP 16 BIT RANDOM SEED
                BNE   RNDOK
                INC   RANDH
RNDOK           LDA   SLTTYPS+3     ;WHAT CARD IS IN SLOT 3?
                CMP   #3            ;IS IT A COM CARD?
                BEQ   COMCK         ;YES, GO CHECK IT
                CMP   #4            ;IS IT A SERIAL CARD?
                BEQ   JDONCK        ;YES, IT CANT BE TESTED
                CMP   #6
                BEQ   FIRMCK
TSTKBD          LDA   ØCØØØ        ;TEST APPLE KEYBOARD
                BPL   JDONCK        ;NO CHARACTER AVAILABLE
                STA   ØCØ1Ø         ;CLEAR KEYBOARD STROBE
                AND   #Ø7F          ;MASK OFF TOP BIT
                TAX
                LDA   SPCHAR        ;See if checking for Apple special characters
                ROR   A             ;is turned off.
                BCS   NOTFOLP2      ;Jump if so
                TXA
                CMP   #11.          ;CTRL-K?
                BNE   NOTK
                LDA   #Ø5B          ;YES, REPLACE WITH LEFT SQR BRACKETT
NOTK            CMP   #1            ;CTRL-A?
                BNE   NTTAB
                JSR   HTAB          ;YES,TAB NEXT MULT 4Ø
                LDA   CONFLGS
                AND   #ØFE
                STA   CONFLGS       ;CLEAR AUTO-FOLLOW BIT
                JMP   DONECK
```

```
NTTAB       CMP   #26.          ;CTRL-Z?
            BNE   NOTFOL        ;NO,PUT CHARACTER IN BUFFER
            LDA   CONFLGS
            ORA   #1
            STA   CONFLGS       ;SET AUTO-FOLLOW BIT
            BNE   DONECK        ;BR ALWAYS
COMCK       LDA   ØCØBE         ;CHARACTER AVAILABLE?
            LSR   A
            BCC   DONECK        ;NO CHARACTER AVAILABLE
            LDA   ØCØBF         ;GET CHARACTER FROM UART
GOTCHAR     AND   #Ø7F          ;MASK OFF BIT 7
NOTFOL      TAX
            LDA   SPCHAR        ;See if console special character checking is
                                ;turned off.
            ROR   A
NOTFOLP2    ROR   A
            BCS   NFMI1         ;Jump if so
            TXA
            LDY   #Ø55
            CMP   (SYSCOM),Y    ;STOP CHAR?
            BNE   NOTSTOP
            LDA   CONFLGS
            EOR   #Ø8Ø
            STA   CONFLGS       ;YES, TOGGLE STOP BIT (BIT 7)
JDONCK      JMP   DONECK
FIRMCK      LDA   #1
            LDY   #Ø3Ø
            JSR   FIRMSTATUS
            BCC   DONECK
            JSR   FREAD1
            JMP   GOTCHAR
NOTSTOP     DEY
            CMP   (SYSCOM),Y
            BNE   NOTBRK
            LDA   CONFLGS
            AND   #Ø3F
            STA   CONFLGS       ;CLEAR FLUSH AND STOP BITS
            .IF   RUNTIME=Ø
            JMP   TOBREAK
            .ELSE
            JMP   @BREAK        ;BREAK OUT
            .ENDC
NOTBRK      DEY
            CMP   (SYSCOM),Y    ;FLUSH?
            BNE   NOTFLUS
            LDA   CONFLGS
            EOR   #Ø4Ø
            STA   CONFLGS       ;TOGGLE FLUSH BIT (BIT 6)
            JMP   DONECK
NFMI1       TXA
```

```
NOTFLUS     LDX   WPTR
            JSR   BUMP
            CPX   RPTR          ;BUFFER FULL?
            BNE   BUFOK
            JSR   BELL
            JMP   DONECK        ;BEEP AND IGNORE CHAR
BUFOK       STX   WPTR
            STA   CONBUF,X      ;PUT CHARACTER IN BUFFER
DONECK      BIT   CONFLGS       ;IS STOP FLAG SET?
            BPL   CKEXIT
            JMP   RNDINC        ;LOOP IF IN STOP MODE
CKEXIT      PLA
            TAY
            PLA
            TAX
            PLA
            PLP
            RTS                 ;ELSE RESTORE STAT AND ALL REG AND RETURN
BUMP        INX                 ;BUMP BUFFER POINTER WITH WRAP-AROUND
            CPX   #CBUFLEN
            BNE   BMPRTS
            LDX   #Ø
BMPRTS      RTS

            ;---------------------------------
            ;
            ; INITIALIZE CONSOLE:
            ;
            ;---------------------------------

CINIT       PLA
            STA   TEMP1         ;SAVE RETURN ADDR
            PLA
            STA   TEMP2
            PLA
            STA   SYSCOM        ;SAVE PTR TO SYSCOM AREA
            PLA
            STA   SYSCOM+1
            PLA
            STA   BREAK         ;SAVE BREAK ADDRESS
            PLA
            STA   BREAK+1
            LDA   TEMP2
            PHA                 ;RESTORE RETURN ADDR
            LDA   TEMP1
            PHA
            LDA   RPTR          ;FLUSH TYPE-AHEAD BUFFER
            STA   WPTR
            LDA   CONFLGS
            AND   #Ø3E
            STA   CONFLGS       ;CLEAR STOP, FLUSH, AUTO-FOLLOW BITS
            JSR   TAB3          ;NO,HORIZONTAL SHIFT FULL LEFT
CINIT2      LDX   #Ø            ;CLEAR IORESULT
            RTS                 ;AND RETURN
```

```
            ;--------------------------------
            ;
            ; READ FROM CONSOLE:
            ; KEYBOARD, COM OR SERIAL CARD IN SLOT 3
            ;
            ;--------------------------------

CREAD       JSR   ADJUST       ;HORIZONTAL SCROLL IF NECESSARY
            LDY   #Ø3Ø         ;SLOT 3
            LDA   SLTTYPS+3    ;WHAT TYPE OF CARD?
            CMP   #4           ;IS IT A SERIAL CARD?
            BNE   CREAD2       ;NO, CONTINUE
            JSR   RSER         ;YES, READ IT
            AND   #7F          ;MASK OFF TOP BIT
            RTS
CREAD2      JSR   CONCK        ;TEST CHARACTER
            LDX   RPTR
            CPX   WPTR
            BEQ   CREAD        ;LOOP TILL SOMETHING IN BUFFER
            JSR   BUMP
            STX   RPTR         ;BUMP READ POINTER
            LDA   CONBUF,X     ;GET CHARACTER FROM BUFFER
            LDX   #Ø           ;CLEAR IORESULT
            RTS                ;AND RETURN TO PASCAL

            ;--------------------------------
            ;
            ; INITIALIZE PRINTER:
            ; PRINTER IS ALWAYS IN SLOT 1
            ; IT MAY BE A PRINTER,COM,OR SERIAL CARD
            ;
            ;--------------------------------

PINIT       LDY   #Ø1Ø         ;SLOT 1 ; Ø1Ø
            LDA   SLTTYPS+1    ;WHAT CARD IN SLOT 1?
            CMP   #5           ;PRINTER CARD?
            BEQ   CLRIO1       ;YES,NO INIT NEEDED
GENIT       CMP   #4           ;SERIAL CARD?
            BEQ   ISER         ;YES, INIT SER CARD
            CMP   #3           ;COM CARD?
            BEQ   ICOM         ;YES,INIT COM CARD
            CMP   #6
            BEQ   FIRMINIT
            LDX   #9           ;NONE OF ABOVE, OFFLINE
            RTS
```

```
FIRMINIT    PHA
            JSR   SER1
            LDY   #ØD
FVEC1       LDA   (TEMP1),Y
            STA   TEMP1
            LDY   6F8
            PLA
            JMP   @TEMP1

            ;---------------------------------
            ;
            ; INITIALIZE REMOTE:
            ; REMOTE IS ALWAYS IN SLOT 2
            ; IT MAY BE A COM OR SERIAL CARD
            ;
            ;---------------------------------

RINIT       LDA   SLTTYPS+2   ;WHAT CARD IN SLOT 2?
            LDY   #Ø2Ø
            BNE   GENIT       ;BRANCH ALWAYS TAKEN

            ;---------------------------------
            ;
            ; INIT COM CARD, Y=ØNØ
            ;
            ;---------------------------------

ICOM        LDA   #3          ;MASTER INIT
            STA   ØCØ8E,Y     ;TO STATUS
            LDA   #21.
            STA   ØCØ8E,Y     ;SET BAUD ETC
CLRIO1      LDX   #Ø          ;CLEAR IORESULT
            RTS               ;AND RETURN

            ;---------------------------------
            ;
            ; INIT SERIAL CARD, Y=ØNØ
            ;
            ;---------------------------------

ISER        JSR   SER1        ;ASSORTED GARBAGE
            JSR   ØC8ØØ       ;SET UP SLOT DEPENDENTS
CLRIO3      LDX   #Ø          ;CLEAR IORESULT
            RTS               ;AND RETURN
```

```
                ;---------------------------------
                ;
                ; ASSORTED SERIAL CARD SET-UP
                ;
                ;---------------------------------

SER1            STY   Ø6F8         ;STORE NØ
                TYA
                LSR   A
                LSR   A
                LSR   A
                LSR   A
                ORA   #ØCØ
                TAX               ;MAKE ØCN IN X
                LDA   #Ø
                STA   TEMP1
                STX   TEMP2       ;SET UP INDIRECT ADDRESS
                LDA   ØCFFF       ;TURN OFF ALL C8 ROMS
                LDA   (TEMP1),Y   ;SELECT C8 BANK
                RTS

                ;---------------------------------
                ;
                ; WRITE TO CONSOLE:
                ; VIDEO SCREEN, COM OR SERIAL CARD IN SLOT 3
                ;
                ;---------------------------------

CWRITE          JSR   CONCK       ;CONSOLE CHARACTER AVAILABLE?
                BIT   CONFLGS     ;IS FLUSH FLAG SET?
                BVS   CLRIO       ;YES,DISCARD CHARACTER AND RETURN
                TAX               ;SAVE CHARACTER IN X
                LDY   #Ø3Ø        ;SLOT 3 ; Ø1Ø
                LDA   SLTTYPS+3   ;WHAT KIND OF CARD?
                CMP   #3          ;COM CARD?
                BEQ   WCOM        ;YES, WRITE TO COM CARD SLOT 3
                CMP   #4          ;SERIAL CARD?
                BEQ   WSER        ;YES, WRITE TO SERIAL CARD SLOT 3
                CMP   #6
                BEQ   WFIRM
                TXA               ;ELSE RESTORE CHARACTER AND SEND TO SCREEN
VIDOUT          STA   TEMP1       ;SAVE CHARACTER FOR LATER
                JSR   INVERT      ;REMOVE CURSOR
                LDY   CH
                JSR   VOUT2       ;DO THE BUSINESS
                JSR   INVERT      ;RESTORE THE CURSOR
CLRIO           LDX   #Ø          ;CLEAR IORESULT
                RTS               ;RETURN FROM VIDOUT
```

```
WFIRM       TXA
            PHA
            LDA   #Ø
            JSR   IOWAIT
            JSR   SER1
            LDY   #ØF
            JMP   FVEC1

            ;---------------------------------
            ;
            ; WRITE TO SERIAL CARD, Y=ØNØ, CHAR IN X
            ;
            ;---------------------------------

WSER        JSR   CONCK       ;CONSOLE CHARACTER?
            TXA
            PHA               ;SAVE CHARACTER ON STACK
            JSR   SER1        ;ASSORTED GARBAGE
            PLA
            STA   Ø5B8,X      ;SET UP DATA BYTE
            JSR   ØC9AA       ;SEND IT (SHOUT)
            LDX   #Ø
            RTS

            ;---------------------------------
            ;
            ; WRITE TO REMOTE:, CHARACTER IN A
            ;
            ;---------------------------------

RWRITE      TAX               ;SAVE CHARACTER
            LDA   SLTTYPS+2   ;WHAT CARD IN SLOT 2?
            LDY   #Ø2Ø
            BNE   GENW2       ;BRANCH ALWAYS TAKEN

            ;---------------------------------
            ;
            ; WRITE TO PRINTER CARD SLOT1, CHARACTER IN X
            ;
            ;---------------------------------

WPRN        JSR   CONCK       ;CONSOLE CHARACTER AVAILABLE?
            LDA   ØC1C1       ;TEST PRINTER READY
            BMI   WPRN        ;LOOP TILL READY
            STX   ØC Ø9Ø      ;SEND CHARACTER
CLRIO2      LDX   #Ø
            RTS
```

```
                ;---------------------------------
                ;
                ; WRITE TO COM CARD, Y=ØNØ, CHARACTER IN X
                ;
                ;---------------------------------

WCOM            JSR   CONCK        ;CONSOLE CHARACTER?
                LDA   ØCØ8E,Y      ;TEST UART STATUS
                AND   #2           ;READY?
                BEQ   WCOM         ;NO, WAIT TILL READY
                TXA
                STA   ØCØ8F,Y      ;SEND CHARACTER
                LDX   #Ø
                RTS

                ;---------------------------------
                ;
                ; WRITE TO PRINTER:, CHARACTER IN A
                ;
                ;---------------------------------

PWRITE          TAX                ;SAVE CHARACTER IN X
                LDA   LFFLAG       ;TEST LINE-FEED FLAG
                BPL   LFPASS       ;PASS IF BIT7=Ø
                CPX   #1Ø.         ;IS IT A LINE-FEED?
                BEQ   CLRIO        ;YES, IGNORE
LFPASS          LDY   #Ø1Ø         ;SLOT 1
                LDA   SLTTYPS+1    ;WHAT KIND OF CARD?
GENW            CMP   #5           ;PRINTER CARD?
                BEQ   WPRN         ;YES, WRITE TO PRINTER CARD
GENW2           CMPL  #4           ;SERIAL CARD?
                BEQ   WSER         ;YES, WRITE TO SERIAL CARD
                CMP   #3           ;COM CARD?
                BEQ   WCOM         ;YES, WRITE TO COM CARD
                CMP   #6
                BEQ   WFIRM
OFFLINE         LDX   #9
                RTS

                ;---------------------------------
                ;
                ; READ FROM REMOTE:
                ;
                ;---------------------------------

RREAD           LDA   SLTTYPS+2    ;WHAT CARD IN SLOT 2?
                LDY   #Ø2Ø
GENR            CMP   #4           ;SERIAL CARD?
                BEQ   RSER         ;GET FROM SERIAL CARD
                CMP   #3           ;COM CARD?
                BEQ   RCOM         ;GET FROM COM CARD
                CMP   #6
                BEQ   RFIRM
                BNE   OFFLINE      ;CARD NOT RECOGNIZED
```

```
            ;---------------------------------
            ;
            ; READ FROM COM CARD, Y=NØ
            ;
            ;---------------------------------

RCOM        JSR   CONCK        ;CHECK FOR CONSOLE CHARACTER
            LDA   ØC Ø8E,Y     ;TEST UART STATUS
            LSR   A            ;TEST BIT Ø
            BCC   RCOM         ;WAIT FOR CHARACTER
            LDA   ØC Ø8F,Y     ;GET CHARACTER
            LDX   #Ø
            RTS
RFIRM       LDA   #1
            JSR   IOWAIT
FREAD1      JSR   SER1
            PHA
            LDY   #ØE
            JMP   FVEC1

            ;---------------------------------
            ;
            ; READ FROM SERIAL CARD, Y=ØNØ
            ;
            ;---------------------------------

RSER        JSR   CONCK        ;CONSOLE CHARACTER AVAILABLE?
            JSR   SER1         ;ASSORTED GARBAGE
            JSR   ØC84D        ;GET A BYTE (SHIFTIN)
            LDA   Ø5B8,X       ;GET BYTE Ø678+SLOT
            LDX   #Ø
            RTS

FIRMSTATUS  PHA
            JSR   SER1
            LDY   #1Ø
            JMP   FVEC1

IOWAIT      JSR   CONCK
            PHA
            JSR   FIRMSTATUS
            PLA
            BCC   IOWAIT
            RTS
```

MERGING DRIVERS FOR SYSTEM.ATTACH

This program will allow you merge several driver files into one ATTACH file
if needed.

```
(*$I-*)
PROGRAM ADMERG;

(*  This program is short and sweet and not very robust.  It's that
    way to make it easy to type in.  Now that you know the record size of
    ATTACH.DATA files, you are welcome to make it more robust yourself.  *)

  TYPE DEVREC = ARRAY [0..8] OF INTEGER;

  VAR  IDNAME, ODNAME:          STRING;
       CH:                      CHAR;
       INDATA, OUTDATA:         FILE OF DEVREC;

  PROCEDURE ERROR (ST: STRING);         (* Any error aborts the program *)
  BEGIN
     WRITELN ('ERROR => ', ST);
     WRITE   ('Press RETURN to exit ADMERG: ');
     READLN;
     EXIT (ADMERG);
  END;

BEGIN
   WRITE ('Name of Data file you will create: ');
   READLN (ODNAME);
   REWRITE (OUTDATA, ODNAME);
   IF IORESULT <> 0 THEN ERROR ('opening Output file.');
   CH := 'Y';
   WHILE (CH = 'Y') OR (CH = 'y') DO BEGIN
      WRITE ('Name of Input file: ');
      READLN (IDNAME);
      RESET (INDATA, IDNAME);
      IF IORESULT <> 0 THEN ERROR ('opening Input file.');
      WHILE NOT EOF (INDATA) DO BEGIN
          OUTDATA^ := INDATA^;
          PUT (OUTDATA);
          GET (INDATA)
      END;
      CLOSE (INDATA);
      WRITE ('Another Input file?  Y)es, N)o: ');
      READLN (CH)
   END;
   CLOSE (OUTDATA, LOCK)
END.
```

Apple II Applications Software

APPLE POST AND DOS 3.3

Apple Post can be MUFFINed successfully to the 16-sector format. However, Apple Post does not take advantage of the extra disk space and the number of entries per list remains the same as for 13-sector diskettes. The advantage of using 16-sector format is convenience in booting up the Apple Post program.

APPLE POST AND PRINTERS

Apple Post doesn't work with some printers because Apple Post looks at the PROMs on the interface card too see which kind it is, and ignores non-Apple cards. You can force the decision by adding the following modifications to two of the modules:

UTILITY MODULE	and	OUTPUT MODULE
13135 V= <code>		1125 X2= <code>

where <code> = 1 for an Apple Parallel Printer Interface
2 for an Apple High Speed Serial Interface
3 for an Apple Communications Interface

Suggestion: Use the code for the serial card (2) when interfacing a non-Apple card so that Apple Post will use the firmware on the interface card. Be sure to make backup copies of your diskettes before making ANY changes!

LABELS AND THE LEFT MARGIN IN APPLE POST

To move the printing on the LABELS away from the left margin, change one line in the OUTPUT MODULE:

37430 PRINT BF$(W) TO 37430 PRINT TAB (10);BF$(W)

ZIP CODES IN APPLE POST

In LIST or PHONELIST, an error will occur if CITYSTATE=20 and SORTKEY=10 and ZIP>=6 digits (sum > 36). The status module will come on line with an error. To avoid this problem add the following line to the OUTPUT MODULE:

35123 IF LEN (R3$(3))+ LEN (R$(4)) + LEN (R$(7)) > 36 THEN G3$="":GOTO 35130

The above fix will become increasing important when zip codes become 9 digits long.

APPLE POST FILE SIZE

Apple Post is limited to 500 names per diskette in a 2 disk system because the boot drive must hold the Apple Post program. If the file has more than 500 names, you have two options:

* get another drive for each additional 500 names
* break the file into logical parts, each with a unique name, using separate diskettes

APPLE POST AND THE SAMPLE LIST

The Sample data file in Apple Post is for example purposes ONLY! It has room for only a few names - If you used this file as a mailing list file you must re-enter the data into a new file. See NEWLIST on page 9 of the Applepost manual. The warning against using the SAMPLE file as a mailing list file is located on page 5 of the Applepost manual.

FORMAT STATEMENT IN APPLE POST

The FORMAT command allows an Applepost user to set the label height for printing labels. Setting the label height to 5 lines will print 6 lines per label nevertheless. To correct this problem add the following line to the OUTPUT MODULE:

```
37448 IF LH=5 THEN 37465
```

APPLE PLOT AND VISICALC FILE CONVERTER

The Visicalc file converter has a problem as it is sent on the diskette. We are working on a solution to the problem.

MODIFYING THE APPLE PLOT PROGRAM

Apple Plot is not easy to modify because there is a hole in the middle of it and Applesoft will normally ignore the portion of the program behind the hole. You have to re-connect the two parts of the program before you can change anything and re-split it when you're done. This will require further research to make sure that there are no hidden problems.

QUME GRAPHICS DRIVER FOR APPLE PLOT

The Qume graphics dump routine in Apple Plot starts at 16458.

APPLE PLOT WITH OTHER GRAPHICS PRINTERS

Using printers other than the Silentype or Qume with the Apple Plot program is accomplished by saving the graphs generated by the user in Apple Plot in picture files (page 18-20 of the Apple Plot manual), exiting Apple Plot by pressing Q, and then BLOADing the picture files into memory and dumping the high resolution graphics page to your printer. A graphics driver must exist for the printer you are using. As an example, the Computer Station has one for the IDS 440G printer. If such a graphics driver is not available and the printer has graphics capabilities, a suitable graphics driver must be written for the printer in question.

APPLEPLOT AND DOS 3.3

The MUFFIN program can be successfully run on the Apple Plot diskette with no side effects whatsoever. See Appendix K of the DOS 3.3 manual for instructions on the use of the MUFFIN program .

An ERROR #53 (ILLEGAL QUANTITY ERROR: page 81, Applesoft II reference manual) can occur in Apple Plot for two known reasons. The first reason for the occurrence of the ILLEGAL QUANTITY ERROR is the floating labels setable by the user in the X-axis parameters menu. Page 13 thru 15 of the Appleplot manual describes the X-axis parameters menu format. If the user responds to the inquiry ´USE LABEL #1 ?´ or ´USE LABEL #2 ?´ with a ´Yes´ response and then responds ´No´ to the ´CHANGE LABEL #1 ?´ or ´CHANGE LABEL #2 ?´ inquiry which follows, the result will be an ERROR #53 during the graph display.

The second reason is connected with the bar chart graph format. There is a bug in the bar chart format that makes the ending plot point of the plot range (selected in the edit/entry menu; Apple Plot manual page 32) equal to the number of X-axis divisions (see page 13 of the Apple Plot manual). The change in the plot range will occur during the display of the graph and if the ending plot point becomes less than the beginning plot point an ERROR #53 will occur.

Another problem connected with the changing of the plot range´s ending plot point to equal the X-axis divisions is that the number of bars actually plotted may be less than the user expects. An example of how this can occur is illustrated below.

If the plot range is set up as shown below:

```
STARTING POINT 5
ENDING   POINT 10
```

And the X-axis parameters are as shown below:

```
NUMBER OF X-AXIS DIVISIONS 5
MAXIMUM X-AXIS VALUE       10
MINIMUM X-AXIS VALUE       5
```

The displayed graph will be one bar plotted at the far right of the screen corresponding to point number 10. Upon examining the plot range values the ENDING POINT is now set to 5 and the result is one bar plotted. If the number of X-axis divisions was 6 (MINIMUM X-AXIS VALUE 4) then the ENDING POINT of the plot range would have been set to 6 during the display and only 2 bars on the right side of the screen would have been plotted. This symptom is exhibited only in the bar chart format. Other graph formats are unaffected.

RECEIVING OPTIONS WITH THE PORTFOLIO EVALUATOR

1. Boot DOS 3.2.1 master diskette
2. Copy Portfolio Evaluator to a blank diskette
3. Load FETCH from new copy
4. Type in:

```
6290 I = I + ( SGN( I) + (I = 0)) * VAL( MID$( I$,J-2,2) / VAL(
     MID$( I$,J+1)): RETURN
7230 I$ = "-"
```

5. UNLOCK FETCH
6. SAVE FETCH
7. LOCK FETCH

This new copy will be able to retrieve options but not quotes.

NOTES:

The change to line 6290 in the FETCH module corrects a bug in which all stocks or options with values of less than one dollar are reported as "NO DATA". Use it whenever this problem occurs.

By changing line 7230 to set I$ equal to different symbols, quotes other than stocks or options can be received (see your Dow Jones manual). Only securities with the same output format as stocks and options will be received correctly.

CORRECTIONS TO PORTFOLIO EVALUATOR

The following changes must be made in the DISPLAY program. To fix the ex-dividend bug which causes a large net loss to be reported in the evaluation display when a stock goes ex-dividend, change the following lines to read:

```
7485 H1=11:H2=19:I=X(6,S): GOSUB 880

7490 H1=20:H2=28:VI=0:I=ABS(X(4,S)): IF I <> 0 THEN VI=I: GOSUB 880: GOTO
     7500

7510 H1=29:H2=39:DG=100:I=V: IF I<>0 THEN GOSUB 800: GOTO 7250

7644 G=ABS(X(4,S)): IF G=0 THEN G=ABS(X(0,S)): IF G=0 THEN H1=22:H2=32:
     I$="NO DATA":GOSUB 825:PRINT:GOTO 7677

7650 G=( ABS(G) - ABS(X(6,S)) )* ABS(X(7,S)):TG=TG+G
```

See "Receiving Options with the Portfolio Evaluator" for another correction.

PORTFOLIO EVALUATOR AND THE MICROMODEM

To allow the Portfolio Evaluator to run with the D.C. Hayes Micromodem II in slot #3, make the following changes:

FIRST! You must make the changes indicated by D.C. Hayes to the LOGIN program.

Now, modify their modifications thusly:

```
5142 IF LGIN=-1 THEN PRINT D$;"PR#3":PRINT Z$
5196 PRINT D$;"PR#3":PRINT Z$:GOTO 6000
5197 PRINT D$;"PR#3"
5202 PRINT D$;"PR#3":PRINT Z$
5645 PRINT D$;"PR#3"
```

And add these new ones:

```
5710 ST=49342: REM ADDR OF UART STATUS SLOT #3
5705 POKE 835,190:POKE 844,191: REM MODIFY COM22.OBJ
```

Make these changes to the FETCH program:

```
7150 PRINT:PRINT D$;"PR#3"
7900 ST=49342: REM UART STATUS SLOT #3
```

DOW JONES NEWS AND QUOTES REPORTER

There is currently no method of accessing the Dow Jones network from DATAPAC (Canada). Canadian customers must dial over the border to a U.S. access number. Portfolio Evaluator could be easily modified to allow DATAPAC operation, but the News and Quotes Reporter cannot.

The News and Quotes Reporter package will not work reliably with the new 300/1200 baud modems recently installed by Tymnet. An update diskette has been supplied to all dealers to correct this problem.

Version 1.0 of the News and Quotes Reporter will not work with a Communications, Serial or some 80 Column boards in Slot #3. This problem is caused by the PASCAL run-time system. An update diskette has been supplied to your dealer to solve this problem.

APPLEWRITER MODIFICATION FOR LOWER CASE DISPLAY

This note is taken from a letter from Lou Rivas of Canoga Park, California. It allows Applewriter to be used with the Paymar Lower Case Adapter or other similar devices. The few ASCII codes not available on the Apple keyboard are also supported. This modification still uses the normal Applewriter files, only the display routines were changed.

```
]UNLOCK TEDITOR
]BLOAD TEDITOR
]CALL -151

811:8D 10 C0 4C 48 18
AE6:20 64
AE8:20 6B 18
1848:C9 81 F0 01 60 AD 00 C0
1850:10 FB C9 AF D0 06 A9 DC
1858:8D 10 C0 60 C9 AD D0 F8
1860:A9 DF D0 F4 20 78 18 91
1868:28 C8 60 C9 A0 90 06 20
1870:01 15 20 78 18 4C F6 FD
1878:C9 E0 90 02 49 40 C9 C0
1880:90 02 09 20 C9 40 B0 08
1888:C9 20 B0 02 09 40 09 80
1890:60
```

Then, to check your typing, enter:

```
811.816 AE6.AE8 1549.154B 1848.1890
```

which should duplicate the above information.

Now save the editor with:

```
]BSAVE TEDITOR, A$803, L$10F8
]LOCK TEDITOR
```

The extra characters are "_", "\" and "|" and may be entered into a file with:

"|"	<CTRL-A> /
"\"	<ESC> <CTRL-A> /
"_"	<ESC> <CTRL-A> -

<CTRL-A> will display a white block on the screen but nothing will be printed.

The total character set is now:

```
  ! " # $ % & ' ( ) * + , - . /   0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O   P Q R S T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k l m n o   p q r s t u v w x y z { | } ~
```

(With some printers, some of the characters may be defined differently.)

ELEMENTARY, MY DEAR APPLE ERRATA
030-0114-00

Page 2
It is stated that when a blinking cursor and a prompt is shown, one can press the RESET key and you will always return to the main menu. There is a possibility that RESET will not return to the menu.

Page 5
In the paragraph beginning " If you press a wrong key...." it says to go back and correct errors using the forward arrow key. It should say the back or left arrow key.

Page 7
There is no mention that in the LEMONADE game you can use the arrow keys to correct mistakes.

One cannot turn off the music in the LEMONADE program.

APPLESOFT TOOL KIT MANUAL ERRATA
030-0113-00

Page 11
If you want your program ... change line 180 to:

```
180 CALL ADRS + 3
```

APPLE 6502 ASSEMBLER/EDITOR ERRATA
030-0112-00

Page 7
"In the following descriptions and examples, the horizontal line" should read "...the vertical line"

Page 17
The end or the first paragraph should read "end of the range." instead of "end of the line."

Page 19
Paragraph 4 starts out "Back up your diskettes" when it means "Back up your files". It also should state that you use a different file name each time you back up your file.

Page 20
The description for TLOAD mentions the appendix "Editor Tape Formats" which doesn't exsist. See the TSAVE command.

Page 20 Paragraph 4
The tape format for TSAVE is different than Basic programs. It is similiar to Integer Basic but not Applesoft. The files can not be loaded directly anyway because of the tokenizing of key words that both Basics do.

Page 20
The TSAVE format should be:
(10-second leader)
(16-bit file length)
(10-second leader)
(file data)

Page 28, paragraph 6
"Commands Available During Assembly" actually appear on page 30 as "Assembly Mode Commands"

Page 30
The section, "Assembly Mode Commands", doesn't mention that the escape key will stop the listing like the space bar and switch the display the right half of the virtual 80-column display. The space bar is used to reset the display to the left half.

Page 31
The second paragraph should start "The Listing OFF Command" instead of "Listing ON Command"

Page 32
The discussion of LABEL mixes the terms "Label" and "Symbol". This should be clarified.

Page 32
The last line refers to the appendix on Object File Formats. This information is on Page 48 and 49 under "Assembler Directive Summary" and "Operation Code Summary".

Page 33
Paragraph 2 line 6 refers to the syntax summary on page 49 when the information is on page 48.

Page 60

Sector	Byte (Hex)	Contents of byte
1 to N	6 to c1+5	Binary code image, of length in bytes 4 and 5 above
	c1+6	Begin Relocation Dictionary, which consists of N 4 bytes entries N is variable (0 to ?)

THE SHELL GAMES MANUAL ERRATA
030-0066-00

Page 11
Item 4 mentions that the listing of ´Animals and Their Young´ is on page 9 when it´s really on page 8.

Page 26
Paragraph 3 mentions that problem 5 resides from 9025 to 9029 when it is actually from 9030 to 9034.

Page 34
Modifying Professor True states that you are given 5 strings while you really only get 4, D1$ to D4$.

Page 35-36
You must save the program and re-run it before any flag changes will take effect.

TAX PLANNER MANUAL ERRATA
030-0171-00

Page 49
Item VI reads:

```
LET
    Excess Itemized Deductions = class 2 deductions
            -.6 * (Adj gross income - class 1 deductions)
```

Which should read:

```
LET
    Excess Itemized Deductions = class 1 deductions
            -.6 * (Adj gross income - class 2 deductions)
```

Apple II Hardware

(Section empty)

Apple II Peripherals

TABBING WITH APPLE PERIPHERALS

This driver allows the user to TAB normally without substituting POKE 36,X for TAB (X). It does require that the PR#(slot) command be replaced with a CALL wherever it occurs in the program and that DOS 3.3 is in the system. It is customized for a certain type of interface in a certain slot when it is entered and will not work in any other configuration.

SOFTWARE ENTRY

First you must determine the value for the four variable parameters for the card. Look at the table for the interface that you will be using and determine the values A, B, C and D for the slot in which the interface will be used.

PARALLEL INTERFACE

SLOT	1	2	3	4	5	6	7
A	02	02	02	02	02	02	02
B	C1	C2	C3	C4	C5	C6	C7
C	F9	FA	FB	FC	FD	FE	FF
D	07	07	07	07	07	07	07

SERIAL INTERFACE

SLOT	1	2	3	4	5	6	7
A	07	07	07	07	07	07	07
B	C1	C2	C3	C4	C5	C6	C7
C	F9	FA	FB	FC	FD	FE	FF
D	05	05	05	05	05	05	05

SILENTYPE PRINTER

SLOT	1	2	3	4	5	6	7
A	07	07	07	07	07	07	07
B	C1	C2	C3	C4	C5	C6	C7
C	04	04	04	04	04	04	04
D	CF	CF	CF	CF	CF	CF	CF

COMMUNICATIONS CARD PRINT ROUTINE

A = 07 B = 03 C = 02 D = 03

Enter the monitor with CALL - 155 and, use the values from the above tables for the items within brackets "< >", type:

```
3B0:A9 <SLOT>
:20 95 FE
:A9 8D
:20 ED FD
:A9 C5
:85 36
:A9 03
:85 37
:4C EA 03
:20 <A> <B>
:48
:AD <C> <D>
:85 24
:68
:60
```

To check your typing, enter:

```
3B0L
```

and compare your listing to the one below for a parallel interface in slot 1.

```
03B0- A9 01     LDA #$01
03B2- 20 95 FE  JSR $FE95
03B5- A9 8D     LDA #$8D
03B7- 20 ED FD  JSR $FDED
03BA- A9 C5     LDA #$C5
03BC- 85 36     STA $36
03BE- A9 03     LDA #$03
03C0- 85 37     STA $37
03C2- 4C EA 03  JMP $03EA
03C5- 20 02 C1  JSR $C102
03C8- 48        PHA
03C9- AD F9 07  LDA $07F9
03CC- 85 24     STA $24
03CE- 68        PLA
03CF- 60        RTS
```

Now return to BASIC with 3D0G.

SAVING THE PROGRAM TO DISK

The driver should be in memory before the printer is used. Save the driver by typing:

```
BSAVE TABBER, A$3B0, L$20
```

USING THE PRINTER

NOTE: If the Apples video is enabled, DO NOT TAB past the 40th column as doing so may ruin the program in memory.

The first time you want to use the printer you must load the driver. From command mode type:

```
BLOAD TABBER
```

This may be done from within a running program by entering:

```
100 PRINT D$;"BLOAD TABBER"
```

assuming that D$ contains a Control-D.

The first time the printer is needed, the interface must be initailised. This may be done with CALL 944 to turn on the printer and send out a carriage return to initialise the interface. Subsequently, initialization is not needed and CALL 954 will return to the printer without printing a carriage return.

Don't use PR#, use CALL 944 or CALL 954 instead!!

Most of these interfaces have parameters that can be POKEd to modify the interface and the Parallel Printer Interface has its Ctrl-I commands. These POKEs and commands should be issued just after the initial CALL 944. They needn't be repeated after CALL 954.

When you want to switch back to the video monitor for output, type PR#0. From within a program this must be in the form of:

```
200 PRINT D$;"PR#0"
```

EXAMPLE PROGRAM:

```
100  LET D$ = CHR$ (4)
110  PRINT D$;"BLOAD TABBER"
120  CALL 944
130  PRINT "THIS WILL BE PRINTED ON THE PRINTER"
140  PRINT D$;"PR#0"
150  PRINT "AND NOW BACK TO THE SCREEN"
160  CALL 954
170  PRINT "NOW FOR A TABBING DEMO"
180  FOR J = 1 TO 76 STEP 5
190  PRINT TAB (J); J;
200  NEXT J
210  PRINT
220  PRINT D$;"PR#0"
230  END
```

INITIALIZING APPLE PERIPHERALS WITH POKES

Neither the PR# or IN# commands in Applesoft or Integer BASIC initialize the interface to which they refer. This can cause problems for the user who needs to modify the parameters of the interface for his application because he must send a character through the interface before poking in the new parameters. The following lists are the POKEs needed to initialize the memory locations used by the various interfaces. Please refer to the manual for the interface you will be using for more information on what each POKE will do.

Included for each interface is a list of POKEs that will replace the PR# and IN# commands in your programs. These POKEs muse be used to reap the benefits of the previous pokes. The CALL 1002 should be used if you will be using DOS commands while the interface is enabled. However, if speed is of the essence, don't use the CALL 1002 until after the data transfer has been made since DOS does slow down I/O. These POKEs must all be on one command line separated by colons to work from command mode. They may have separate line numbers in a program.

The normal way to reset the I/O to the Apple video and the keyboard is:

```
D$ = "": REM CTRL-D
PRINT D$;"PR#0"
PRINT D$;"IN#0"
```

However, this will only work after a PRINT statement and will be ignored after a GET or PRINT terminated with a comma or semicolon. To avoid having to do the extra PRINT you can use:

```
CALL -375 : REM  THIS IS IN#0
CALL -365 : REM  THIS IS PR#0
CALL 1002 : REM  THIS RECONNECTS DOS
```

SPECIAL NOTE: All of these interfaces have the option of echoing to the Apple video while processing output and your program or variables will suffer if you don't disable the video output while printing past the 40th column.

PARALLEL PRINTER INTERFACE

```
POKE 1400 + <SLOT>,80          Carriage width
POKE 1656,0                    Character counter
POKE 1784 + <SLOT>,137         Set command prefix to CTRL-I
```

The following are for the Centronics card only:

```
POKE 1912 + <SLOT>,0          No video, no linefeed
 or               ,1          No video, enable linefeed
 or               ,128        Enable video, no linefeed
 or               ,129        Enable video, enable linefeed

POKE 54,2                     PR#<SLOT>
POKE 55,192 + <SLOT>
CALL 1002
```

COMMUNICATIONS INTERFACE

```
POKE 1784 + <SLOT>,32         Lower case (page 17)
POKE 1912 + <SLOT>,0          Video echo (page 17)
POKE 2040 + <SLOT>,17         STAT       (page 27)
POKE -16242 + <SLOT> * 16,3   Reset ACIA (page 27)
POKE -16242 + <SLOT> * 16,17  Status     (page 27)

POKE 54,5                     PR#<SLOT>
POKE 55,192 + <SLOT>
POKE 56,7                     IN#<SLOT>
POKE 57,192 + <SLOT>
CALL 1002
```

SERIAL INTERFACE CARD

```
POKE 1144 + <SLOT>,64         BRATE      (page 21)
POKE 1272 + <SLOT>,2          STBITS     (page 21)
POKE 1400 + <SLOT>,7          STATUS     (page 22)
POKE 1528 + <SLOT>,0          Character counter
POKE 1784 + <SLOT>,80         PWDTH      (page 23)
POKE 1912 + <SLOT>,9          NBITS      (page 23)
POKE 2040 + <SLOT>,129        FLAGS      (page 24)

POKE 54,7                     PR#<SLOT>
POKE 55,192 + <SLOT>
POKE 56,5                     IN#<SLOT>
POKE 57,192 + <SLOT>
CALL 1002
```

CARRIAGE RETURN DELAY

This program introduces a variable delay after a carriage return. The delay gives slower printers time to return the print head to the left margin without dropping characters. DOS 3.3 is required to use this modification.

First you must decide which slot the interface will go in and enter the delay program. The program is customized for this slot number and won't work if used with a different configuration. Enter the program using the values from the table for words in brackets, < >.

SLOT	1	2	3	4	5	6	7
CODE	C1	C2	C3	C4	C5	C6	C7

	PARALLEL	COMMUNICATIONS	SERIAL
TYPE	02	05	07

DELAY: The delay is measured in tenths of a second and is entered in hexadecimal format. Hence, "5" is 1/2 second and "A" is one second. The usual default value is "5".

Enter the monitor with CALL -155 and enter the following:

```
390:A9 <SLOT>
:20 95 FE
:A9 80
:20 ED FD
:A9 A5
:85 36
:A9 03
:85 37
:4C EA 03
:20 <TYPE> <CODE>
:C9 8D
:D0 0E
:AD BB 03
:48
:A9 C2
:20 A8 FC
:68
:E9 01
:D0 F5
:60
:<DELAY>
```

To check your typing, enter:

```
390L 3BB
```

and compare your listing to the one shown below for Slot #1 and a Parallel Printer Interface.

```
3090-  A9 01      LDA #$01
0392-  20 95 FE   JSR $FE95
0395-  A9 80      LDA #$80
0397-  20 ED FD   JSR $FDED
039A-  A9 A5      LDA #$A5
039C-  85 36      STA $36
039E-  A9 03      LDA #$03
03A0-  85 37      STA $37
03A2-  4C EA 03   JMP $03EA
03A5-  20 02 C1   JSR $C102
03A8-  C9 8D      CMP #$8D
03AA-  D0 0E      BNE $03BA
03AC-  AD BB 03   LDA $03BB
03AF-  48         PHA
03B0-  A9 C2      LDA #$C2
03B2-  20 A8 FC   JSR $FCA8
03B5-  68         PLA
03B6-  E9 01      SBC #$01
03B8-  D0 F5      BNE $03AF
03BA-  60         RTS
03BB-  05
```

Now return to BASIC using 3D0G.

The program must be in memory before the printer can be used with the delay. Save the program by typing:

```
BSAVE CR DELAY, A$390, L$2C
```

USING THE PRINTER

The first time you want to use the printer you must load the program and initialize the interface. From command mode type:

```
BLOAD CR DELAY
CALL 912
```

This may be done from within a program by entering:

```
100 PRINT D$;"BLOAD CR DELAY" : CALL 912
```

assuming that D$ is a Control-D.

If you want to switch back to the video monitor for output type "PR#0", or from within a program; "200 PRINT D$;"PR#0".

Then, to reconnect the printer, type "CALL 922" from command mode or "300 CALL 922" from within your program.

NOTES:

If the delay needs to be adjusted from BASIC, the command used to accomplish this is POKE 955,<DELAY>. The delay, in this case, is in decimal.

HIGH SPEED SERIAL INTERFACE MANUAL ERRATA
030-0012-00

Page 16
Switch 4 is reversed... ON = delay enabled
OFF = delay disabled

ADDENDUM TO THE HIGH SPEED SERIAL INTERFACE MANUAL ERRATA
031-0012-00

Note:
The P8A PROM interferes with the operation monitor routines that use $3C. The P8A uses this location as a temp and doesn't restore it.

COMMUNICATIONS INTERFACE MANUAL ERRATA
030-0008-01

Page 17
The lower case info only applies to TERMINAL modes and the method to use it isn't obvious. Also, the POKE 1784+n,160 causes the lower case to be displayed as FLASHING. POKE 1784+n,224 will cause lower case to be INVERSE.

Page 24
DATAMOVER will not work with Applesoft. It will require a complete re-write of the program.

Page 31
The Com Card Print Routine was written when the only available DOS was version 3.1. POKE 845,110 will fix it to work with DOS 3.2 or DOS 3.3.

Page 36
The modification for 4800 baud in the manual is missing a wire. Add a wire from pin 15 of A1 to pin 15 of A2. Chip A1 is still not required.

PARALLEL PRINTER INTERFACE MANUAL ERRATA
030-0005-01

Page 18
Notes: For users of Applesoft BASIC...
Applesoft allows the ´PR#´ command. The rest of the note is still valid but of limited utility.

Page 18
Using Printer Commands in BASIC Programs:

```
>10 PR#1        Turns off Printer Card
  should read
>10 PR#1        Turns on Printer Card
```

GENERAL NOTES

TAB does not work properly with this interface. The Integer Basic version is limited to 40 columns and an Applesoft TAB(20) will sometimes output 20 instead of going to column 20. Use POKE 36,T (where T is the tab value). This works with either Basic.

Entering the control-I printer commands from Basic command level will give a SYNTAX ERROR. It doesn´t hurt anything but be warned.

CENTRONICS INTERFACE CARD
030-0005-01 (Parallel Printer Interface)

Page 17
Command Explanation

Ctrl-I I RETURN has no effect. Use control-I 0 instead
Ctrl-I K RETURN has no effect. This version never generates a line feed

Page 18 Notes: For users of Applesoft BASIC...
Applesoft allows the ´PR#´ command. The rest of the note is still valid but of limited utility.

Page 18
Using Printer Commands in BASIC Programs

```
>10 PRINT PR#1      Turns off Printer Card.   should be
>10 PRINT PR#1      Turns on Printer Card

>30 PRINT "^I I" should be
>30 PRINT "^I 0"

>40 PRINT "^I K"  does not turn off the Line Feed as there never was one.
```

Page 19
Example Of Control From a BASIC Program

```
(20 PRINT "^IK";)  is never needed or valid

>50 PRINT "^II";   should be replaced with
>50 PRINT "^IO";
```

Page 19
Listing Programs Containing Print Commands

```
>^IK  is never needed or valid
```

GENERAL NOTES

TAB does not work properly with this interface. The Integer Basic version is limited to 40 columns and an Applesoft TAB(20) will sometimes output 20 instead of going to column 20. Use POKE 36,T (where T is the tab value). This works with either Basic.

Entering the control-I printer commands from Basic command level will give a SYNTAX ERROR. It doesn´t hurt anything but be warned.

SILENTYPE MANUAL ERRATA
030-0095-00

Page 11
This procedure for printing a HI-RES image will not work. When you go to the Filer, Pascal clears the HI-RES buffer. I suggest that the program be modified:

```
  PROGRAM SPIRO;

  USES TURTLEGRAPHICS, APPLESTUFF;
  VAR ANGLE, DISTANCE : INTEGER;
      CH:CHAR;                          (* ADDED *)
  .
  .
  .
      ANGLE := ANGLE + 5;
    END;
    CH:=CHR(17);                        (* ADDED *)
    UNITWRITE(6,CH,1,0,12);             (* ADDED *)
    TEXTMODE
  END.
```

Page 38
Line 4, ´for the right margin is 2´ should be ´for the left margin is 2´.

Page 39
Program line 6040 should end with THEN GOTO 6070

GRAPHICS TABLET MANUAL ERRATA
030-0076-00

Page 26
Line 5 starts ´uninitialized´ but should read ´initialized´

Page 55
The following lines are wrong:

2475 GOTO 2340 should be 2475 GOTO 2350

2490 GOTO 2340 should be 2490 GOTO 2348

Page 55
Line 2390 near the bottom of the page is wrong.

Apple ///
System Software

APPLE /// BUSINESS BASIC

Business Basic uses the same floating point math package as Applesoft but it only displays 6 digits when it prints. This limitation was built in because Applesoft's math package will develop errors in the last three decimal places. If more than 6 digits is required then routines to do floating point arithmetic using long integers must be written and implemented.

PRINTING FROM APPLE /// BUSINESS BASIC

To print out to the RS-232 serial port type the following:

```
OPEN #1,".PRINTER"    (use .SILENTYPE for Silentype printer)
OUTPUT #1
CATALOG
```

The catalog will go out to the printer.

APPLE /// SYSTEM CONFIGURATION PROGRAM

The System Configuration Procedure in the Apple /// utilities diskette cannot be used with a 96K Apple ///. The reason for this is that the drivers to be changed are loaded into memory and there is no room for the drivers plus the System Configuration utilities. The solution at the current time is to use the utilities diskette with a 128K Apple ///.

PRINTING FROM APPLE /// EMULATION MODE

This routine will send data out to a printer connected to the RS-232 port on the back of the Apple ///. It will handshake the same as Apple /// native mode and also send out a linefeed after a carriage return. The normal way to use it from a Basic program is to BLOAD the routine from the disk and CALL 768. The entry point for using it with Apple Writer is $319. The replacement value for x, y, and z are in the Apple /// Standard Device Drivers Manual on pages 82 and 83:

x = the low digit from the table on page 83 (parity)
y = the high digit from the table on page 83 (word length)
z = the low digit from the table on page 82 (speed)

```
300:A9 19              LDA #$19        ;SET UP I/O VECTORS
302:85 36              STA $36
304:A9 03              LDA #$03
306:85 37              STA $37
308:20 EA 03           JSR $03EA
30B:8D F1 C0           STA $C0F1       ;RESET ACIA
30E:A9 2B              LDA #$2B        ;$xB, default=odd parity
310:8D F2 C0           STA $C0F2
313:A9 2E              LDA #$2E        ;$yz, 7 data bits, 1200 baud
315:8D F3 C0           STA $C0F3
318:60                 RTS
319:48          ENTER  PHA
31A:AD F1 C0    LOOP   LDA $C0F1
31D:49 10              EOR #$10
31F:20 70              AND #$70
321:D0 F7              BNE LOOP
323:68                 PLA
324:8D F0 C0           STA $C0F0
327:C9 8D              CMP #$8D
329:D0 07              BNE OUT
32B:48                 PHA
32C:A9 8A              LDA #$8A
32E:20 17 03           JSR ENTER
331:68                 PLA
332:60          OUT    RTS
```

Apple ///
Applications
Software

PRINTING FROM VISICALC ///

Apple /// Visicalc is the only portion of the Information Analyst which does not follow the naming conventions for SOS drivers. The RS-232 driver in Visicalc is named ".QUME", and the default ".PRINTER" refers to the Apple Silentype printer. To print to the RS-232 serial port, type the following:

P (for Print), F (for File), .QUME (for the RS-232 driver)

VISICALC /// COLUMN INSERTIONS

If column 254 has be used in a given session, Visicalc assumes that there is no more room and won't allow column insertions. The easiest fix is to save the file to disk and re-load it. This will reset Visicalc's pointer.

Apple ///
Hardware

APPLE /// EMULATION to APPLE II HARDWARE COMPARISON

This section covers the differences between the Apple II hardware and the emulation mode found on the Apple ///. To use this information, you need to have a copy of the Apple II Reference manual. The following pages represent the DIFFERENCES which would exist if the manual were written specifically for Apple /// emulation mode. Chapter numbers, Titles, and Figure numbers are such that they correspond to the existing manual.

CHAPTER 1

APPROACHING YOUR APPLE

THE KEYBOARD

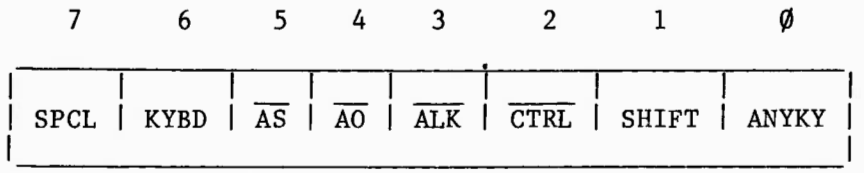
The Apple II Emulation Keyboard

Number of Keys:	75
Coding:	Upper Case ASCII
Number of Codes:	1Ø2 (All ASCII except lower case alpha)
Output:	2 Ports A - Seven bits, plus strobe
	B - Modifier keys
Rollover:	2 key
Special keys:	CONTROL
	ESCAPE
	RESET
	TAB
	ALPHA LOCK
	<left arrow>
	<right arrow>
	<up arrow>
	<down arrow>
	<open apple>
	<closed apple>

READING THE KEYBOARD

Keyboard data A and Clear Keyboard Strobe are identical to the Apple II keyboard. Keyboard port B is new and it allows reading the special keys on on the keyboard.

Keyboard port B is organized as :



SPCL — 1 - A key is pressed on the numeric pad, cursor keys, space bar, TAB, ESCAPE
Ø - The last key pressed was not one of the above keys.

KYBD — 1 - "ON" light of the keyboard active
Ø - "ON" light burned out, or Keyboard not connected

$\overline{AS}$ — 1 - Solid Apple key not pressed with an ASCII key since a Clear Keyboard Strobe
Ø - Solid Apple key pressed with an ASCII key

$\overline{AO}$ — 1 - Open Apple key not pressed now
Ø - Open Apple key pressed now

$\overline{ALK}$ — 1 - APLHA LOCK key up
Ø - ALPHA LOCK key down

$\overline{CTRL}$ — 1 - CONTROL key not pressed now
Ø - CONTROL key pressed now

SHIFT — 1 - SHIFT key pressed with an ASCII character
Ø - SHIFT key not pressed with last ASCII character or Clear Keyboard Strobe

ANYKY — 1 - ASCII character key is pressed now
Ø - No ASCII character key is pressed now

The RESET button is located on the upper edge of the keyboard between the "\|" key and the "7" on the numeric pad. In Apple II emulation mode it generates an NMI (Non Maskable Interrupt) to the 65Ø2 microprocessor. This is not the same as a RESET to the microprocessor chip which actually powers the chip down. Thus illegal operation codes which cause the micro

to hang cannot be escaped from through the use of this key alone. However, the RESET pin (31) on the slots 1 through 4 will go low for the duration of the button pressing.

In emulation mode the NMI will normally jump to the RESET code which is in the AUTOSTART monitor. Thus in most cases there is no difference.

Pressing CONTROL and RESET together will cause a true RESET to the 65Ø2 and the computer will return to Apple /// native mode with the disk attempting to boot an Apple /// disk.

The REPT key is not on the keyboard as each ASCII key has auto repeat which repeats approxiametly 11 times a second. The repeat speed can be raised to 3Ø characters a second by pressing the Solid Apple key after pressing the character key.

High speed repeat is also available on the cursor keys by pressing the key firmly.

READING THE KEYBOARD

Table 1 : Keyboard Special Locations

Location: Hex	Decimal		Description
$CØØØ	49152	-16384	Keyboard Port A
$CØØ8	4916Ø	-16376	Keyboard Port B
$CØ1Ø	49168	-16368	Clear Keyboard Strobe

Table 2: Keys and their Associated ASCII Codes (Bit 7 always set)

Key	Alone	CONTROL	SHIFT	Both
<space>	$AØ	$AØ	$AØ	$AØ
ESCAPE	$9B	$9B	$9B	$9B
1!	$B1	$B1	$A1	$A1
2@	$B2	$B2	$CØ	$8Ø
3#	$B3	$B3	$A3	$A3
4$	$B4	$B4	$A4	$A4
5%	$B5	$B5	$A5	$A5
6^	$B6	$B6	$DE	$9E
7&	$B7	$B7	$A6	$A6
8*	$B8	$B8	$AA	$AA
9(	$B9	$B9	$A8	$A8
Ø)	$BØ	$BØ	$A9	$A9
-_	$AD	$AD	$DF	$9F
=+	$BD	$BD	$AB	$AB
\\|	$DC	$9C	$FC	$FF
TAB	$89	$89	$89	$89
[{	$DB	$9B	$FB	$9B
]}	$DD	$9D	$FD	$9D
;"	$A7	$A7	$A2	$A2
RETURN	$8D	$8D	$8D	$8D
,<	$AC	$AC	$BC	$BC
.>	$AE	$AE	$BE	$BE
/?	$AF	$AF	$BF	$BF
<left arrow>	$88	$88	$88	$88
<right arrow>	$95	$95	$95	$95
<up arrow>	$8B	$8B	$8B	$8B
<down arrow>	$8A	$8A	$8A	$8A
.	$AE	$AE	$AE	$AE
-	$AD	$AD	$AD	$AD
ENTER	$8D	$8D	$8D	$8D
A	$C1	$81	$C1	$81
B	$C2	$82	$C2	$82
C	$C3	$83	$C3	$83
D	$C4	$84	$C4	$84
E	$C5	$85	$C5	$85
F	$C6	$86	$C6	$86
G	$C7	$87	$C7	$87
H	$C8	$88	$C8	$88
I	$C9	$89	$C9	$89
J	$CA	$8A	$CA	$8A
K	$CB	$8B	$CB	$8B
L	$CC	$8C	$CC	$8C
M	$CD	$8D	$CD	$8D
N	$CE	$8E	$CE	$8E
O	$CF	$8F	$CF	$8F
P	$DØ	$9Ø	$DØ	$9Ø
Q	$D1	$91	$D1	$91
R	$D2	$92	$D2	$92
S	$D3	$93	$D3	$93
T	$D4	$94	$D4	$94

U	$D5	$95	$D5	$95
V	$D6	$96	$D6	$96
W	$D7	$97	$D7	$97
X	$D8	$98	$D8	$98
Y	$D9	$99	$D9	$99
Z	$DA	$9A	$DA	#9A

THE VIDEO CONNECTOR

The Apple /// has several connections for video output.

The B/W VIDEO connector is located in the center of the back connector panel between the COLOR VIDEO and AUDIO connectors. It provides black and white positive composite synchronization video signal. This can be plugged directly into most black and white monitors with an input impedance of 75 Ohms. An RCA-type male to male connection cable is provided for this connection.

The COLOR VIDEO CONNECTOR provides outputs for NTSC compatible black and white, NTSC color, and RGB. An M&R Enterprises SUP´R´MOD can be attached to the color video output to create a channel 33 color video signal for use with a normal television. This same company products a board to allow the connection of an RGB monitor. A complete explanation of this connector is in the Apple /// Owner´s Manual.

THE HIGH-RESOLUTION GRAPHICS (HI-RES) MODE

The Apple II emulation mode high resolution graphics are identical to the Apple II except some combinations of colors on the right edge of the screen will cause the left edge pixels to blink. This is normal, although distracting.

THE SPEAKER

The speaker function is identical to the Apple II with the following additional features.

A reference to location 49216 (or the equivalent addresses -16336 or hexadecimal $CØ4Ø) will cause a Ø.1 second 1 KHz tone to be produced which is similar to the sound the AUTOSTART monitor makes when the BELL character is sent to the screen. The advantage to this is Ø.1 seconds of CPU time is returned to the user, since only 1 microsecond is required to start the BELL sound.

The AUDIO connector at the back of the Apple /// provides the same signal as the speaker. When you insert a miniature phone-tip plug into this jack, the Apple's internal speaker is silenced; if there is an amplifier or other device properly connected to the plug, then that device will receive all audio signals generated by the Apple. The signal is a Ø.5 volt peak-to-peak audio signal on its tip and signal ground on its ring.

THE CASSETTE INTERFACE

The cassette interface is completely eliminated on the Apple ///. References to the cassette output port at 49184 (or the equivalent -16352 of hexadecimal $CØ2Ø) will cause pin 39 of the I/O slots to go low for a microsecond. This is for use by Apple /// native mode peripherals to deselect to $C8ØØ ROM address space.

Reading the cassette input port at 49248 or the equivalents -16288 or hexadecimal $CØ6Ø will read joystick switch Ø into bit 7.

THE GAME I/O CONNECTOR

The Game I/O connector does not exist in the Apple ///.

ANNUNCIATOR OUTPUTS

The annunciator outputs do not exist in the Apple ///. These locations address the A/D convertors to read the joysticks.

ONE-BIT INPUTS

The one-bit inputs are used to read the switches on the joysticks. There are three one-bit inputs. They have the addresses 49248 through 49251 (-16288 through 16285 or hexadecimal $CØ6Ø through $CØ63).

The input switches are read in bit 7 of their data bytes. Port B reads input switches 1 and 3. Port A reads input switches Ø and 2.

Table 1Ø: Input / Output Special Locations

Function	Address: Decimal	Hex	Read/Write
Speaker	49200 -16336	$CØ3Ø	R/W
Beep	49216 -1632Ø	$CØ4Ø	R/W
Deselect $C8ØØ for Apple /// peripherals (pin 39 in slots)	49184 -16352	$CØ2Ø	R/W
Joystick switch Ø	49248 -16288	$CØ6Ø	R(bit 7)
Joystick switch 1	49249 -16287	$CØ61	R(bit 7)
Joystick switch 2	4925Ø -16286	$CØ62	R(bit 7)
Joystick switch 3	49251 -16285	$CØ63	R(bit 7)
A/D Select Ø	4924Ø -16296	$CØ58	R/W
A/D select Ø	49241 -16295	$CØ59	R/W
A/D Select 1	49246 -1629Ø	$CØ5E	R/W
A/D Select 1	49247 -16289	$CØ5F	R/W
A/D Select 2	49242 -16294	$CØ5A	R/W
A/D Select 2	49243 -16293	$CØ5B	R/W
A/D Ramp charge	49244 -16292	$CØ5C	R/W
A/D Start timeout	49245 -16291	$CØ5D	R/W
A/D Timeout	49254 -16282	$CØ66	R(bit 7)
Clock millisecond counter ($NØ)	49264 -16272	$CØ7Ø	R(bits 7-4)

Table 9: A/D Selection

A/D 2	A/D 1	A/D Ø	Input
Ø	Ø	Ø	Ground
Ø	Ø	1	Joystick, Port B, X axis
Ø	1	Ø	Joystick, Port B, Y axis
Ø	1	1	Joystick, Port A, X axis
1	Ø	Ø	Joystick, Port A, Y axis
1	Ø	1	Clock Battery
1	1	Ø	No connection
1	1	1	Reference Voltage

ANALOG INPUTS

The system has two joystick ports with provisions for two A/D inputs each. Port A reads A/D inputs 3 and 4 while Port B reads inputs 1 and 2.

To read the A/D inputs, the software must select the desired input and charge the ramp capacitor for at least 5ØØ microseconds. Then the ramp is started and the time measured until the A/D timeout goes low. The discharge time is proportional to the input voltage.

STROBE OUTPUT

The strobe output has been replaced by Ø.1 second 1 Khz tone from the speaker.

AUTOSTART ROM / MONITOR ROM

The Apple II emulation only comes with a modified version of the Autostart ROM. This is a write protected RAM which is loaded at Apple II emulation boot time.

CHAPTER 5

INPUT / OUTPUT STRUCTURE

BUILT-IN I/O

DATA INPUTS

The keyboard is read through two ports which provide the complete state of the Apple /// keyboard. Port A is identical in both location and content to the Keyboard Data Input of the Apple II.

FLAG INPUTS

The method of reading the flags is the same though some flags, such as the cassette input, are not present.

STROBE OUTPUTS

The Utility strobe doesn't exist in emulaton mode and the Game strobe is in a different location.

TOGGLE SWITCHES

Only the speaker toggle is present.

SOFT SWITCHES

A/D selection locations are soft switches which produce a 3 bit value to determine which device is to be attached to the analog input. The Annunciator outputs are not present.

Table 22: Built-In I/O Locatons

	$Ø	$1	$2	$3	$4	$5	$6	$7
$CØØØ	Keyboard Port A Input							
$CØØ8	Keyboard Port B Input							
$CØ1Ø	Clear Keyboard Strobe							
$CØ2Ø	Deselect all expansion I/O space (pin 39) for Apple /// cards							
$CØ3Ø	Speaker Toggle (lus) pulse							
$CØ4Ø	Speaker Beep (1 KHz for Ø.1 second)							
$CØ5Ø	gr	tx	nomix	mix	pri	sec	lores	hires
$CØ58	$\overline{\text{A/D Ø}}$	A/D Ø	$\overline{\text{A/D 2}}$	A/D 2	A/D CHG	A/D ST	$\overline{\text{A/D 1}}$	A/D 1
$CØ6Ø	SWØ	SW 1	SW 2	SW 3	$\overline{\text{IRQ 2}}$	$\overline{\text{IRQ 1}}$	$\overline{\text{A/D TM}}$	MUX1
$CØ7Ø	Clock millisecond output ($NØ)							
$CØ9Ø-$CØ9F	Slot 1 Device Select (pin 41) goes low during C1M							
$CØAØ-$CØAF	Slot 2 Device Select (pin 41) goes low during C1M							
$CØBØ-$CØBF	Slot 3 Device Select (pin 41) goes low during C1M							
$CØCØ-$CØCF	Slot 4 Device Select (pin 41) goes low during C1M							
$CØEØ	$\overline{\text{Disk Stepper Motor Phase A}}$							
$CØE1	Disk Stepper Motor Phase A							
$CØE2	$\overline{\text{Disk Stepper Motor Phase B}}$							
$CØE3	Disk Stepper Motor Phase B							
$CØE4	$\overline{\text{Disk Stepper Motor Phase C}}$							
$CØE5	Disk Stepper Motor Phase C							
$CØE6	$\overline{\text{Disk Stepper Motor Phase D}}$							
$CØE7	Disk Stepper Motor Phase D							

$CØE8 Disk motor off

$CØE9 Disk motor on

$CØEA Select Drive 1 (Built-in)

$CØEB Select Drive 2 (First external)

$CØEC Q6L

$CØED Q6H

$CØEE Q7L

$CØEF Q7H

$CØFØ ACIA Receive/Transmit Data register

$CØF1 ACIA Status register

$CØF2 ACIA Command register

$CØF3 ACIA Control register

$C1ØØ-$C1FF Slot 1 I/O Select (Pin 1) goes low during C1M low

$C2ØØ-$C2FF Slot 2 I/O Select (Pin 1) goes low during C1M low

$C3ØØ-$C3FF Slot 3 I/O Select (Pin 1) goes low during C1M low

$C4ØØ-$C4FF Slot 4 I/O Select (Pin 1) goes low during C1M low

CHAPTER 6

HARDWARE CONFIGURATION

THE MICROPROCESSOR

The only difference is the 65Ø2B microprocessor is used in the Apple ///. The B designation indicates it can use a faster clock frequency, but Emulation mode uses the standard 1.Ø23 MHz clock.

SYSTEM TIMING

System timing is identical.

POWER SUPPLY

Input voltage:	1ØØ VAC to 125 VAC, 5Ø/6Ø Hz 1ØØW max. 2ØØ VAC to 25Ø VAC, 5Ø,6Ø Hz 1ØØW max.
Supply voltages:	+5.Ø VDC -2%/+3% +11.8 VDC +6%/-6% -5.Ø VDC +6%/-1Ø% -12.Ø VDC +1Ø%/-1Ø%
Power Consumption:	1ØØ W max.
Full load power output:	+12V: 2.5A + 5V: 4.ØA - 5V: Ø.25A -12V: Ø.3ØA

PERIPHERAL BOARD I/O

The Apple /// implements only slots 1 through 4. Slot 6 is always a disk interface card and slots 5 and 7 contain either a SERIAL or COMMUNICATIONS card. Slot Ø scratchpad RAM exists but no provision is made to put a LANGUAGE card or FIRMWARE card into the system. Thus the RAM is limited to 48K with a 12K ROM chosen at boot time.

PERIPHERAL CARD I/O SPACE

Slot 6 I/O space $CØEØ-$CØEF contains the hardware for the disk interface. Slot 7 I/O space $CØFØ-$CØF3 contains the addresses for the onboard ACIA.

PERIPHERAL CARD ROM SPACE

Slot 5 and slot 7 contain code which is functionally equivalent to the COMMUNICATIONS or SERIAL card for the Apple II. They differ in that they use the built-in ACIA. For a more complete explanation, see "SERIAL AND COMMUNICATONS CARD EMULATION ".

Slot 6 contains a copy of the Apple II 16 sector Boot PROM.

ROM MEMORY

The Applesoft, Integer Basic, and Autostart Monitor "ROMS" are actually write protected RAMs in the Apple ///. When the Emulation mode disk is booted, it loads RAM memory with an image of each set of ROMs. Whichever language is selected when the Apple II disk is booted is loaded into the address space ($DØØØ-$FFFF) and write protected.

RAM MEMORY

In Emulation mode there is always 48K of RAM. It is addressed $ØØØØ to $BFFF. There is no provision for a slot Ø Language or Firmware card.

"USER 1" JUMPER

There is no "User 1" jumper in the Apple ///.

THE GAME I/O CONNECTOR

There is no Game I/O connector in the Apple ///.

THE KEYBOARD

The keyboard is different in design but the locations of the Keyboard Data Input and the Clear Keyboard Strobe are the same. For more information, see "THE KEYBOARD" in chapter 1.

CASSETTE INTERFACE JACKS

There are no cassette interface jacks in the Apple ///.

POWER CONNECTOR

The power connector is different but is not user accessible.

SPEAKER

The speaker is identical to the Apple II.

PERIPHERAL CONNECTORS

The Apple II emulation redefines a few of the pins on the connector and adds several new ones.

The most significant difference is that interrupts will not be sent to the 65Ø2 from the slots. In fact the IRQ (pin 3Ø) is an input to the card so the card can't even determine if an interrupt is occurring. Thus Emulation mode runs without interrupts, period.

The RES (pin 31) is an output to the card and goes low when the RESET key is pressed on the keyboard. However the microprocessor is actually performing an NMI, not a RESET.

Figure 21. Peripheral Connector Pinout

GND	26	25	+5V
DMAOK	27	24	NOT USED
$\overline{\text{DMAI}}$	28	23	NOT USED
$\overline{\text{IONMI}}$	29	22	$\overline{\text{TSADE}}$ (Open collector)
$\overline{\text{IRQ}}$	3Ø	21	RDY (Open collector)
$\overline{\text{IORES}}$	31	2Ø	$\overline{\text{I/O STROBE}}$
$\overline{\text{INH}}$	32	19	PHØ
-12V	33	18	R/$\overline{\text{W}}$
-5V	34	17	A15
SYNC	35	16	A14
C7M	36	15	A13
Q3	37	14	A12
$\overline{\text{C1M}}$	38	13	A11
$\overline{\text{IOCLR}}$	39	12	A1Ø
C1M	4Ø	11	A9
$\overline{\text{DEV SEL}}$	41	1Ø	A8
D7	42	9	A7
D6	43	8	A6
D5	44	7	A5
D4	45	6	A4
D3	46	5	A3
D2	47	4	A2
D1	48	3	A1
DØ	49	2	AØ
+12V	5Ø	1	$\overline{\text{I/O SELECT}}$

Table 33: Peripheral Connector Signal Description

Pin:	Name:	Description:
1	$\overline{\text{I/O SELECT}}$	This line, normally high, will become low when the microprocessor references page $Cn, where n is the individual slot number. This signal become active during PHØ (nominally 5ØØns) and will drive 12 LSTTL loads.
2-17	AØ-A15	The buffered address bus. The address on these lines becomes valid within 3ØØns after the beginning of $\overline{\text{C1M}}$ and remains vaild through PHØ. These lines will each drive 8 LSTTL loads.
18	R/$\overline{\text{W}}$	Buffered Read/$\overline{\text{Write}}$ signal. This becomes valid at the same time the address bus does, and goes high during a read cycle and low during a write. This line can drive up to 1Ø LSTTL loads.
19	PHØ	A 1 MHz signal which is identical to C1M. This line will drive 5 LSTTL inputs.
2Ø	$\overline{\text{I/O STROBE}}$	This line will go low during C1M when the address bus contains an address between $CØØØ and $CFFF. This line will drive 12 LSTTL loads.
21	RDY	The 65Ø2's RDY input. This line should change only during C1M, and when low will halt the microprocessor on the next read cycle. This line has a 1K ohm pullup to +5V. This line should be driven from an open collector output.
22	$\overline{\text{TSADB}}$	A low on this line from the peripheral will cause the address bus to tri-state for Direct Memory Access (DMA) applications. This has a 1 K ohm resistor pullup to +5V. This should be driven from an open collector output.
23		Not used in an Apple ///.
24		Not used in an Apple ///.
25	+5V	Positive 5-volt supply, 2.Ø amps total for all peripheral boards together (but note a limit of 1.5 Watts per board).
26	GND	System circuit ground. Ø volt line from power supply. Do not use for shield ground.

27	DMAOK	Acknowledge signal to the peripheral following its request for the special Direct Memory Access (DMA) mode. Informs the peripheral that the DMA can now proceed.
28	$\overline{\text{DMAI}}$	Direct Memory Access (DMA) interrupt. Requests the Apple /// DMA mode. Has a 1 K ohm pullup to +5. This should be driven from an open collector output.
29	$\overline{\text{IONMI}}$	Input/Output Non-Maskable Interrupt. This is equivalent to the IORES (pin 31) line as it will execute the same code in the Autostart ROM. This line should be driven by an open collector output.
3Ø	$\overline{\text{IRQ}}$	This line is ignored in Apple II emulation mode. It should be driven by a TTL output.
31	$\overline{\text{IORES}}$	Input/Output Reset signal used to reset the peripheral devices. Pulled low by a power on or RESET key. This line will drive 12 LSTTL loads.
32	$\overline{\text{INH}}$	Inhibit line. When a device pulls this line low, all system memory is disabled. This line has a 1 K ohm pullup resistor to +5V and should be driven from an open collector output.
33	-12V	Negative 12 volt supply, 2ØØmA total for all peripheral boards together.
34	-5V	Negative 5 volt supply, 2ØØmA total for all peripheral boards together.
35	SYNC	The 65Ø2 opcode synchronization signal. Can be used for external bus control signals. Will drive 1Ø LSTTL loads.
36	C7M	Seven MHz high frequency clock. Will drive 1Ø LSTTL loads.
37	Q3	A 2MHz (nonsymetrical) general purpose timing signal. Will drive 1Ø LSTTL inputs.
38	$\overline{\text{C1M}}$	Complement of C1M clock. This will drive 12 LSTTL loads.
39	$\overline{\text{IOCLR}}$	Provides the $C8ØØ space disable function directly without address decoding ($CFFF is used for Apple II peripherals. It is addressed from $CØ2x. This line will drive 12 LSTTL loads.

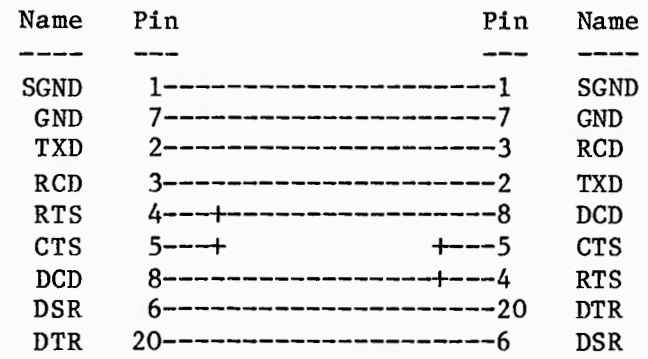
4Ø	$\overline{\text{C1M}}$	Phase C1M clock. This is the same as the microprocessor's 1 MHz clock. This will drive 12 LSTTL loads.
41	$\overline{\text{DEVICE SELECT}}$	This line becomes active (low) on each peripheral connector when the address bus is holding address between $C0n0 and $C0nF where n is the slot number plus $8. This line will drive 12 LSTTL loads.
42-49	D7-D0	The 8-bit system data bus. During a write cycle, data is set up by the 6502 less than 300ns after the beginning of $\overline{\text{C1M}}$. During a read cycle the 6502 expects data to be ready no less than 100ns befor the end of $\overline{\text{C1M}}$. These lines will drive 8 LSTTL inputs.
50	+12V	Positive 12 volt supply, 300mA total for all peripheral boards together.

Apple ///
Peripherals

MODEM ELIMINATOR FOR APPLE ///

One of the built-in devices on the Apple /// is an RS-232 serial interface. This interface can be used in emulation mode or in Apple /// native mode by using the appropriate software driver. However, the interface, as wired, makes the Apple /// a DATA TERMINAL DEVICE. This means that the RS-232 is wired to talk to a MODEM. Similarly, the Qume, and many other RS-232 printers, are also data terminal devices. Although the Qume can physically be plugged in to the Apple ///, the signals are crossed. What is needed is a very simple device called a MODEM ELIMINATOR which flips around a few signals. (It replaces the pair of MODEMs that the computer and terminal are expecting to talk to as well as the communications link.)

The MODEM ELIMINATOR is wired like this:



Other RS-232 printers may require different wiring schemes, depending upon the type of handshake, and pin configuration for the printer.

APPLE /// JOYSTICK INTERFACE

The first release of SOS has a problem with the Joystick interface. It will cause erratic operation when set at low values.

Vendor List

(Section empty)

Bibliography

BIBLIOGRAPHY

BASIC PROGRAMMING/ PROGRAMS

Applesoft Basic Programming Reference Manual, Apple Computer Inc., 1978. Reference manual for Applesoft.

The Applesoft Tutorial, Apple Computer Inc., 1979. Applesoft how-to for beginners.

The Basic Handbook, David A. Lien, Compusoft, 1978.

Basic Programming Manual, Apple Computer Inc., 1978. Integer Basic for beginners.

Mostly Basic: Applications for Your Apple II, Howard Berenbon, Howard W. Sams & Co., 1980.

32 Basic Programs for the Apple Computer, Tom Rugg and Phil Feldman, dilithium Press, 1981. Has color section for Apple.

DISK OPERATING SYSTEM

Do's and Don'ts of DOS, Apple Computer Inc., 1979. Reference for DOS 3.2 and 3.2.1 only.

The DOS Manual, Apple Computer Inc., 1980. New for DOS 3.3 only.

FORTRAN

Apple FORTRAN, Apple Computer Inc., 1980.

FORTRAN 77, Loren P. Meissner and Elliott I. Organick, Addison-Wesley, 1980. FORTRAN 77 full set.

FORTRAN 77: Principles of Programming, Jerrold L. Wagener, John Wiley and Sons, 1980. Full set.

FORTRAN 77 Programming, Walter S. Brainerd, Charles H. Goldberg, Jonathan L. Gross, Harper & Row, 1978. Full set.

Programming Language FORTRAN, ANSI X3.9-1978, American National Standards Institute, 1978. This is the definitive discussion of the full set vs. the subset.

SVS FORTRAN User Reference Manual, Jeffrey Barth and R. Steven Glanville, Silicon Valley Software, 1979. Implementation of the ANSI '77 subset of FORTRAN.

GENERAL REFERENCE

Apple II Reference Manual, Apple Computer Inc., 1979. Main reference for the Apple II and Apple II Plus. Includes monitor listings.

The Apple II User's Guide, Lon Poole, Martin McNiff and Steven Cook, Osborne/McGraw-Hill, 1981.

LISP

Lets Talk LISP, Laurent Siklossy, Prentice-Hall, 1976.

MACHINE LANGUAGE/ MONITOR

Apple Machine Language, Don Inman and Kurt Inman, Reston Publishing Co., 1981. Assembly programming for the Apple. Includes some graphics, I/O specific to the Apple.

The Apple Monitors Peeled, Apple Computer Inc., 1981. List of PEEKs, POKEs and CALLs for both monitor ROMs, explained. Formerly from Rainbow Computing.

Programming the 6502, Rodnay Zaks, Sybex, 1978. Fundamental operation and programming for the 6502 microprocessor.

Synertek Hardware Manual, Synertek, 1976. F0or the 6500 series of microprocessors.

Synertek Programming Manual, Synertek, 1976. Programming for the 6500 series microprocessors.

6502 Software Gourmet Guide and Cookbook, Robert Findley, Scelbi, 1979.

PASCAL

Algorithms + Data Structures = Programs, Niklaus Wirth, Prentice-Hall, 1976. General purpose book on structured programming, with references to PASCAL.

Apple PASCAL Language Reference Manual, Apple Computer Inc., 1980.

Apple PASCAL Operating System Reference Manual, Apple Computer Inc., 1980.

Beginner's Guide for the UCSD PASCAL System, Kenneth L. Bowles, McGraw-Hill, 1980. Has section for Apple. Good all-round book.

BYTE Book of PASCAL, Blaise W. Liffick, ed., BYTE Books, 1979. Collection of articles.

Introduction to PASCAL, Including UCSD PASCAL, Rodnay Zaks, Sybex, 1980. Entry-level textbook.

Introduction to PASCAL, Jim Welsh and John Elder, Prentice-Hall Interational, 1979. Very fundamental, standard PASCAL.

PASCAL, Paul M. Chirlian, Matrix, 1980.

PASCAL: An Introduction to Methodical Programming, William Findlay and David A. Watt, Computer Science Press, 1978. Introductory text for standard PASCAL.

The PASCAL Handbook, Jacques Tiberghien, Sybex, 1981. Excellent reference for reserved words in PASCAL. Includes implementation differences.

PASCAL Primer, David Fox and Mitchell Waite, Howard W. Sams & Co., 1981. UCSD PASCAL, with special notes for Apple users.

A Practical Introduction to PASCAL, I. R. Wilson and A. M. Addyman, Springer-Verlag, 1979. Standard PASCAL.

A Primer on PASCAL, Richard Conway, David Gries and E. C. Zimmerman, Winthrop Publishers, 1976. Standard PASCAL.

Problem Solving Using PASCAL, Kenneth L. Bowles, Springer-Verlag, 1977. Written for the UCSD system, but good beginning reference. Does not cover all facets of the language.

Programming for Poets, Richard Conway, James Archer and Ralph Conway, Winthrop, 1980.

Programming in PASCAL, Peter Grogono, Addison-Wesley, 1978. Excellent reference for standard PASCAL, but applicable to all. Recommended.

Structured Programming and Problem-Solving with PASCAL, Richard B. Kieburtz, Prentice-Hall, 1978.

User Manual and Report, Kathleen Jensen and Niklaus Wirth, Springer-Verlag, 1974. Considered to be the ultimate definition of the PASCAL language, but somewhat difficult to decipher.

PILOT

Apple PILOT Editor's Manual, Apple Computer Inc., 1980. How to use the editors.

Apple PILOT Language Reference Manual, Apple Computer Inc., 1980.

Common PILOT Language Reference Manual, Larry Kheriaty and George Gerhold, Western Washington University, 1978. Apple PILOT was based on this version.

SOFTWARE DIRECTORIES

The Book of Apple Computer Software, 14013 Old Harbor Lane, Suite 312, Marina del Rey, CA 90291. 1981.

Educational Software Directory, Sterling-Swift, 1981.

Purser's Magazine, P.O. Box 466, El Dorado, CA 95623. 1980.

Skarbek's Software Directory, 11990 Dorsett Rd, Maryland Hets, MO 63043. 1980.

WIDL Video, 5245 W. Diversey, Chicago, IL 60639.

1981 Apple II/III Software Directory, Gerald Van Diver and Rolland Love, Vital Information Inc., 350 Union Station, Kansas City, MO 64108, (913) 384-3860. 1981

VENDOR NAMES AND ADDRESSES

ALPHABETICAL LISTING

3-G Co.
Rt. 3, Box 28A, Dept. BT
Gastori, OR 97119
(503) 662-4492

Aardvark Software
783 N. Water St.
Milwaukee, WI 53202
(414) 289-9988

Abacus Software
P.O. Box 7211
Grand Rapids, MI 49510

Abtech
12333 Sunnyvale-Saratoga Rd.
Saratoga, CA 95070
(408) 446-2013

ABW Corp.
P.O. Box M 1047
Ann Arbor, MI 48106
(313) 971-9364

Action-Research Northwest
11442 Marine View Dr. SW
Seattle, WA 98146
(206) 244-9360

Addmaster Corp.
Box 387
416 Junipero Serra Dr.
San Gabriel, CA 91776
(213) 285-1121

Advanced Business Technology
12333 Saratoga-Sunnyvale Rd.
Saratoga, CA 95070
(408) 446-2013

Advanced Logic Systems
491 Macara Ave., Suite 1009
Sunnyvale, CA 94086
(408) 730-0306

Adventure International
P.O. Box 3435
Longwood, FL 32750
(305) 862-6917

Adwar Video Corp.
100 Fifth Ave.
New York, NY 10011
(212) 691-0976

ALF Products
1448 Estes
Denver, CO 80215
(303) 234-0871

Amdek Corp.
2420 East Oakton St., Suite E
Arlington Heights, IL 60005
(312) 364-1180

Andromeda Inc.
P.O. Box 19144
Greensboro, NC 27410
(919) 852-1482

Apparat, Inc.
4401 South Tamarac Parkway
Denver, CO 80237
(303) 741-1778

Application Software Development
504 Lakemead Way
Redwood City, CA 94062
(415) 366-6124

Applied Analytics Inc.
5406 Roblee Dr.
Upper Marlboro, MD 20870

Arizona Computer Systems Inc.
P.O. Box 125
Jerome, AZ 86331
(602) 634-7301

ATV Research
13B Broadway
Dakota City, NE 68731
(402) 987-3771

Avant-Garde Creations
P.O. Box 30161
Eugene, OR 97403
(503) 345-3403

BPI Systems
1600 W. 38th, Suite 444
Austin, TX 78731
(512) 454-2801

BCD Associates
1216 Blackwelder Ave.
Oklahoma City, OK 73106
(405) 524-7403

John Bell Engineering
P.O. Box 338, Dept. 8
Redwood City, CA 94064
(415) 367-1137

Bit-3 Computer Corp.
1890 Huron St.
St. Paul, MN 55113
(612) 926-6997

Broderbund Software
Box 3266
Eugene, OR 97403
(503) 343-9024

BSR Ltd.
Route 303
Blauvelt, NY 10913
(914) 358-6060

BYTE Books
70 Main St.
Peterborough, NH 03458

California Computer Systems
250 Caribbean Dr.
Sunnyvale, CA 94086
(408) 734-5811

Cameo Electronics Inc.
1626 Clementine
Anaheim, CA 92802
(714) 535-1682

Cases, Inc.
P.O. Box 33820
Seattle, WA 98133
(206) 365-5210

Cavri Systems
26 Trumbull St.
New Haven, CT 06511
(203) 562-9873
4979

Charles Mann & Associates
7594 San Remo Trail
Yucca Valley, CA 92284
(714) 365-9718

CJM Industries Inc.
P.O. Box 2367
Reston, VA 22090
(703) 620-2444

Compumax
P.O. Box 1139
Palo Alto, CA 94301
(415) 321-2881

CompuSoCo
26251 Via Roble
P.O. Box 2325
Mission Viejo, CA 92690

CompuServe
Personal Computing Division
5000 Arlington Centre Blvd.
Columbus, OH 43220
(614) 4457-8600

Compu TA
9533 Grossmont Blvd.
La Mesa, CA
(714) 469-3017

CompuTalker
1730 21st St.
Santa Monica, CA 90404
(213) 392-5230

Computer Case Co.
5650 Indian Mound Ct.
Columbus, OH 43213
(614) 868-9464

Computer Mart
P.O. Box 1664
Lake Havasu, AZ 86403
(602) 855-3357

Computer Services Co.
14109 S.E. 168th Street
Renton, WA 98055
(206) 255-7410

Computer Station
12 Crosswoods Plaza
Granite City, IL 62040
(618) 452-1860

Computer Stop
16919 Hawthorne Blvd.
Lawndale, CA 90260
(213) 371-4010

Computer Systems Design Group
3632 Governor Drive
San Diego, CA 92122
(415) 856-1954

CompuTime
P.O. Box 5343
Huntington Beach, CA 92646
(714) 536-5000

Compu-Tron Software Services
1414 South Fairplain Ave.
Whittier, CA 90601
(213) 336-7353

Connecticut Information Systems
218 Huntington Rd.
Bridgeport, CT 06608
(203) 579-0472

Continental Software
12101 Jefferson Blvd.
Culver City, CA 90230
(213) 371-5612

Corvus
2029 O'Toole Ave.
San Jose, CA 95131
(408) 946-7700

Cover Craft
P.O. Box 555
Amherst, NH 03031
(603) 889-6811

CPU Inc.
5161 Atlanta Highway
Montgomery, AL 36109

Creative Computing Software
P.O. Box 789-M
Morristown, NJ 07960
(201) 540-0445

Cyborg Corp.
342 Western Ave.
Boston, MA 92135
(617) 782-9820

Crystal Computer
12215 Murphy Ave.
San Martin, CA 95046
(408) 683-0696

D.J. 'A1' Systems Ltd.
Station Road
Ilminster, Somerset
TA19 9BQ, England
04605 4117

Dakin5
P.O. Box 21187
Denver, CO 80221
(800) 525-0463

Data-Safe Products Inc.
1926 Margaret St.
Philadelphia, PA 19124
(215) 535-3004

Datasoft Inc.
16606 Schoenborn St.
Sepulveda, CA 91343
(213) 894-9154

Decision Master
10428 Westpark
Houston, TX 77042
(800) 231-5768 ex306
(800) 392-2348 in TX

Decision Systems
P.O. Box 13006
Denton, TX 76203

Denver Software Co.
36 Steele St., Suite 19
Denver, CO 80206
(303) 321-4551

Digital Systems Engineering
12503 King's Lake Dr.
Reston, VA 22091

Dockside Computing
P.O. Box 5030
Westlake Village, CA 91362

Dymarc Industries Inc.
7133 Rutherford Rd.
Baltimore, MD 21207
(301) 298-3130

Eastern House Software
3239 Linda Dr.
Winston-Salem, NC 27106
(919) 924-2889
748-8446

Eclectic Corp.
2830 Walnut Hill Ln.
Dallas, TX 75229

Educational Programming Systems
1328 Baur Blvd.
St. Louis, MO 63132
(314) 991-0300

Edu-Comp, Inc.
14109 S.E. 168th Street
Renton, WA 98055
(206) 255-7410

Edu-Ware Services
22222 Sherman Way #102
Canoga Park, CA 91303

Electronic Specialists Inc.
171 S. Main St.
Natick, MA 01760
(617) 655-1532

Electronic Systems
P.O. Box 21638
San Jose, CA 95151
(408) 448-0800

Escon Products Inc.
12919 Alcosta Blvd.
San Ramon, CA 94583
(415) 820-1256

ESP Computer Resources
9 Ash St.
Hollis, NH 03049
(603) 465-7264

Galaxy
Dept. MI5
P.O. Box 22072
San Diego, CA 92122
(714) 452-1072

GNT Automatic Inc.
1560 Trapelo Rd.
Waltham, MA 02154
(617) 890-3305

Hayden Book Co.
50 Essex St.
Rochelle Park, NY 07662

Hayes Microcomputer Products
5835 Peachtree Corners East
Norcross, GA 30092
(404) 449-8791

Hazeltine Corp.
Computer Terminal Equipment
Greenlawn, NY 11740
(516) 549-8800

Health Data Products Inc.
222 E. Anapamu St.
Santa Barbara, CA 93101
(805) 965-4477

Heuristics
1285 Hammerwood Ave.
Sunnyvale, CA 94086
(408) 734-8532

High Sierra Software
5541 Highway 50 East, S.0uite 2A
Carson City, NV 89701
(702) 883-6590

High Technology Inc.
Software Products Division
P.O. Box B-14665
8001 N. Classen Blvd.
Oklahoma City, OK 73113
(405) 840-9900

Highlands Computer Services
1422 S.E. 132nd
Renton, WA 98055
(206) 228-6691

Home Computer Center, Inc.
2927 Virginia Beach Blvd.
Virginia Beach, VA 23452
(804) 340-1977

Houston Instruments
One Houston Square
Austin, TX 78752
(512) 837-2820

Illinois Computer Mart
Route 8, Sweet Corners Plaza
Carbondale, IL 62901

Image Computer Products
615 Academy Dr.
Northbrook, IL 60062
(312) 564-5060

Information Unlimited Software
281 Arlington Ave.
Berkeley, CA 94707
(415) 525-9452

Information Technologies Inc.
2816 Harvard East
Seattle, WA 98102
(206) 325-0430

Innovision
P.O. Box 1317
Los Altos, CA 94022

Instant Software
Peterborough, NH 03458
(603) 924-7296

Intelligent Control Systems, Inc.
P.O. Box 14571
Minneapolis, MN 55414
(612) 699-4342

Interactive Microware Inc.
P.O. Box 771
State College, PA 16801
(814) 238-8294

Interactive Structures
112 Bala Ave.
Box 404
Bala Cynwyd, PA 19004
(215) 667-1713

International Apple Core
P.O. Box 976
Daly City, CA 94017

D.R. Jarvis Computing
1039 Cadiz Dr.
Simi, CA 93065
(805) 526-0151

Jasac+
P.O. Box 7000-287
Palos Verdes Peninsula,
CA 90274
(213) 541-5973

Kinetic Systems Inc.
P.O. Box 4111
Baton Rouge, LA 70821
(504) 383-5369

Lamar Instruments
2107 Artesia Blvd.
Redondo Beach, CA 90278
(213) 374-1673

Lax Computer Products
4728 Manhattan Beach Blvd.
Lawndale, CA 90260
(213) 542-4505

Lazer Systems
P.O. Box 55518
Riverside, CA 92517
(714) 682-5268

Link Systems
1655 26th St.
Santa Monica, CA 90404

Livermore Data Systems Inc.
2050 Research Dr.
Livermore, CA 94550

LJK Enterprises Inc.
P.O. Box 10827
St. Louis, MO 63129
(314) 846-2313

Lobo Drives International
354 South Fairview Ave.
Goleta, CA 93117
(805) 683-1576

M&R Enterprises
P.O. Box 61011
Sunnyvale, CA 94088
(408) 738-3772

Macrotronics
P.O. Box 518(A)
Keyes, CA 95328
(209) 667-2888

Mast Development Co.
2212 E. 12th St.
Davenport, Iowa 52803
(319) 326-0141

Mastertype
P.O. Box 5223
Stanford, CA 94305

Matchless Systems
18444 S. Broadway
Gardena, CA 90248
(213) 327-1010

Mauro Engineering
2220 Pack Trail
Mount Shasta, CA 96067
(916) 926-4406

Med Logic Systems
2860 Walnut Ave., Suite C
Tustin, CA 92680

MFJ Enterprises Inc.
Box 494
Mississippi State, MS 39762
(601) 323-5869

Microcom Inc.
6 Faneuil Hall Marketplace
Boston, MA 02109
(617) 367-6362

Micro Co-op Software
P.O. Box 432
West Chicago, IL 60185
(312) 231-0912

Micro Learningware
P.O. Box 2134
N. Mankato, MN 56001
(507) 625-2205

Micromate Electronics Inc.
2094 Front St.
East Meadow, NY 11554
(516) 794-1072

Micro Mind Inc.
917 Midway
Woodmere, NY 11598
(516) 374-6793

Microproducts
30420 Via Rivera
Rancho Palas Verdes, CA 90274
(213) 541-5131

Micro Products Unlimited
P.O. Box 1525
Arlington, TX 76010
(817) 461-8043

Microsoft Consumer Products
400 108th Ave. NE, Suite 200
Bellevue, WA 98004
(206) 454-1315

Micro Works, Inc.
P.O. Box 1110
Del Mar, CA 92014
(714) 942-2400

Min Microcomputer Software, Inc.
5835-A Peachtree Corners East
Norcross, VA 30092
(404) 447-4322

Monarch Computer Products, Inc.
P.O. Box 4081
New Windsor, NY 12550
(914) 562-3100

Mountain Computer
300 Harvey West Blvd.
Santa Cruz, CA 95060
(408) 429-8600

MR Engineering Co.
4730 W. Addison
Chicago, IL 60641
(312) 286-6606

Muse
7112 Darlington Dr.
Baltimore, MD 21234
(301) 661-8531

Nestar Systems Inc.
2585 East Bayshore Rd.
Palo Alto, CA 94303
(415) 493-2223

Nikrom Technical Products
25 Prospect St.
Leominster, MA 01453

Novation Inc.
18664 Oxnard St.
Tarzana, CA 91356
(213) 996-5060

Omega Research
P.O. Box 479
Linden, CA 95236
(209) 334-6666

Omega Software Systems Inc.
1574 Ives Dairy Rd.
North Miami, FL 33179

Omnico Computer Corp.
3300 Buckeye Rd.
Atlanta, GA 30341
(404) 455-8460

Optimal Technology Inc.
Blue Wood 127
Earlysville, VA 22936
(804) 973-5482

Organic Software
1492 Windsor Way
Livermore, CA 94550
(415) 455-4034

Osborne/McGraw-Hill
630 Bancroft Way, Dept. B13
Berkeley, CA 94710
(415) 548-2805

Passport Designs
P.O. Box 478
La Honda, CA 94020
(415) 747-0614

Paymar, Dan
91 Pioneer Dr.
Durango, CO 81301
(303) 259-3598

Personal Business Systems
4306 Upton Ave., S.
Minneapolis, MN 55410

Personal Software
1330 Bordeaux Dr.
Sunnyvale, CA 94086
(408) 745-7841

Powersoft
P.O. Box 157
Pitman, NJ 08071
(609) 589-5500

Professional Medical Software
3604 Foothill Boulevard
La Crescenta, CA 91214
(213) 248-2884

Programma International
2908 N. Naomi St.
Burbank, CA 91504
(213) 954-0240

Progressive Software
P.O. Box 273
Plymouth Meeting, PA 19462

Prometheus Products Inc.
4509 Thompson Ct.
Fremont, CA 94538
(415) 791-0266

Prosoft
3604 Foothill
La Crescenta, CA
(213) 248-2884

Protean Enterprises
P.O. Box 1284
Olympia, WA 98507
(206) 357-6197

Qkit
P.O. Box 35879
Tucson, AZ 85740
(602) 299-9831

Quality Software
6660 Reseda Blvd., Suite 105
Reseda, CA 91335
(213) 344-6599

Racal-Vadic
222 Caspian Dr.
Sunnyvale, CA 94086
(408) 744-0800

Rainbow Computing Inc.
9719 Reseda Blvd.
Northridge, CA 91324
(213) 349-5560

Rak-Ware
41 Ralph Rd.
West Orange, NJ 07052

RCA
Microcomputer Products
Marketing
New Holland Ave.
Lancaster, PA 17604
(717) 697-7661

Retail Sciences
Suite 700
3 Corporate Sq.
Atlanta, GA 30329
(404) 325-8533

Rochester Data Inc.
3000 Winton Rd., S.
Rochester, NY 14623
(716) 244-7804

RTR Software Inc.
1147 Baltimore Dr., Dept. M1
El Paso, TX 79902
(915) 544-4397

Howard W. Sams & Co.
P.O. Box 7092
4300 W. 62nd St.
Indianapolis, IN 46206

Scan-Tron Corp.
P.O. Box 4273
Burlingame, CA 94010
(415) 697-9651

Sci Tronics
523 S. Clewell St.
P.O. Box 5344
Bethlehem, PA 18015
(215) 868-7220

Select Information Systems
919 Sir Francis Drake Blvd.
Kentfield, CA 94904
(415) 459-4003

Sensational Software
P.O. Box 789-M
Morristown, NJ 07960

Sensible Software
6619 Perham Dr.
West Bloomfield, MI 48033
(313) 399-8877

Sirius Software
2011 Arden Way, #225A
Sacramento, CA 95825
(916) 920-1939

Skarbeks Software Directory
11990 Dorsett Road
Maryland Heights, MO 63043
(314) 567-3291

Softech International Corporation
144 W. 15th Street, Suite #6
North Vancouver, BC
Canada V7M 1R5
(604) 984-0477

Soft CTRL Systems
Box 599
West Milford, NY 07480

Soft Touch
P.O. Box 7200
Costa Mesa, CA 92626

Software Publishing Corp.
P.O. Box 50575
Palo Alto, CA 94303
(415) 368-7598

Software Sorcery Inc.
7927 Jones Branch Dr., Suite 400
McLean, VA 22102
(703) 385-2944

Software Technology
for Computers
P.O. Box 428
Belmont, MA 02178

Sorrento Valley Associates
11722 Sorrento Valley Road
San Diego, CA 92121
(714) 452-0101

Southeastern Software
6414 Derbyshire Dr.
New Orleans, LA 70126
(504) 246-8438

Southwestern Data Systems
P.O. Box 582-C
Santee, CA 92071
(714) 562-3670

Special Delivery Software
Apple Computer Inc.
(408) 744-0630

Spectrum Software
P.O. Box 2084
142 Carlow Dr.
Sunnyvale, CA 94087
(408) 738-4387

SSM Microcomputer Products
2190 Paragon Dr.
San Jose, CA 95131
(408) 946-7400

Stellation Two
P.O. Box 2342
Santa Barbara, CA 93120
(805) 966-1140

Stoneware Micro Computer
Products
1930 Fourth St.
San Rafael, CA 94901
(415) 454-6500

Strategic Simulations Inc.
465 Fairchild Dr., #108
Mountain View, CA 94043

Street Engineering
3152 E. La Palma Ave., Suite C
Anaheim, CA 92806
(714) 632-9950

subLogic Communications Corp.
Box V
Savoy, IL 61874
(217) 359-8482

Sybex
2344 Sixth St., Dept. B31
Berkeley, CA 94710
(415) 848-8233

Sympathetic Software
9531 Telhan Dr.
Huntington Beach, CA 92646

Symtec, Inc.
P.O. Box 462
Farmington, MI 48024
(313) 352-1790

Synergistic Software
5221 - 120th Ave., SE
Bellevue, WA 98006
(206) 641-1917

Syntauri Ltd.
3506 Waverly St.
Palo Alto, CA
(415) 949-1017

System Design Software (SDL)
2612 Artesia Blvd., Suite B
Redondo Beach, CA 90278
(213) 374-4471

Tally
8301 South 180th St.
Kent, WA 98031

Talos Systems Inc.
7419 East Helm Dr.
Scottsdale, AZ 85260
(602) 948-6540

TASA (Touch Activated
Switch Arrays)
2346 Walsh Ave.
Santa Clara, CA 95050
(408) 247-2301

Tecmar, Inc.
23600 Mercantile Rd.
Cleveland, OH 44122
(216) 464-7410

Tekism
ABW Corp.
P.O. Box M 1047
Ann Arbor, MI 48106
(313) 971-9364

Telephone Software Connection
P.O. Box 6548
Torrance, CA 90504
(213) 329-3715

Telesensory Systems
3408 Hillview Ave.
Palo Alto, CA 94304
(415) 493-2626

Tercer Medio
Apartado de Correos 62533
Caracas 1060-A Venezuela

Texcom Engineering Associates
P.O. Box 24472
Houston, TX 77013
(713) 458-3720

Thunderware Inc.
P.O. Box 13322
44 Hermosa Ave.
Oakland, CA 94618
(415) 652-1737

United Software of America
750 3rd Ave.
New York, NY 10017
(212) 682-0347

U.S. Robotics
203 N. Wabash, Suite 1718
Chicago, IL 60601
(312) 346-5650

Van Dusen, Stanley
33170 Little John Dr.
Coarsegold, CA 93614
(209) 683-4793

Velostat
3M Corp.
Data Recording Products
Division
223-5N 3M Center
St. Paul, MN 55101

Versa Computing Inc.
887 Conestoga Circle
Newbury Park, CA 91320
(805) 498-1956

Victor Data Products
3900 N. Rockwell St.
Chicago, IL 60618
(312) 539-8200

Video Marketing Inc.
P.O. Box 339
Warrington, PA 18976
(215) 343-3000

Videx
897 NW Grant Ave.
Corvallis, OR 97330
(503) 758-0521

West Coast Consultants
1775 Lincoln Blvd.
Tracy, CA 95376
(209) 835-1785

West Side Electronics
P.O. Box 636A
Chatsworth, CA 91311
(213) 884-4794

Western Micro Data Enterprises
P.O. Box G33
Postal Station G
Calgary, Alberta
Canada T3A 2W1
(403) 247-1621

Whitney Educational Service
2071 Tenth Ave.
San Francisco, CA 94116
(415) 681-4725

WIDL Video
5245 West Diversey
Chicago, IL 60639
(312) 622-9606

John Wiley & Sons Inc.
605 Third Ave.
New York, NY 10158

XPS Inc.
323 York Rd.
Carlisle, PA 17013

Zeus Computing, A1
P.O. Box 712
Downey, CA 90241

Ziatech
2410 Broad St.
San Luis Obispo, CA 93401
(805) 544-9011

VENDORS LISTED BY PRODUCTS

HARDWARE PRODUCTS

AC Line Filter/Isolator

Dymarc Industries
Electronic Specialists
MFJ Enterprises
Protean Enterprises
RKS Enterprises

Amateur Radio

Macrotronics

Analog Input

California Computer Systems
Computer Technology Associates
Interactive Structures
Tecmar

Analog Output

Interactive Structures

Analog Input/Output

Mountain Computer

Bar Code Reader

Advanced Business Technology

Braille I/O

Telesensory Systems

Cases

Cases, Inc.

Clock/Calendar

California Computer Systems
CompuTime
Intelligent Control Systems
Lax Computer Products
Mountain Computer
SciTronics Inc.
Thunderware
Westside Electronics

Digital Interface

California Computer Systems
Interactive Structures
John Bell Engineering
SSM Microcomputer Products

Digital Output Card

John Bell Engineering

Digitizer/Graphics Tablet

Versa Computing

Disk, Cartridge

Cameo Electronics
Lobo Drives

Disk, 5" Floppy

Lobo Drives
Micro-Sci

Disk, 8" Floppy

Lobo Drives
Matchless Systems
Programma International

Disk, Winchester

Corvus
Konan
Lobo Drives

Disk Controller/DOS

Sorrento Valley Associates

Expansion Card 16K

Andromeda Inc.
Computer Stop
Computer Technology Associates
Microsoft Consumer Products
Prometheus Products
R.H. Electronics
Videx

Expansion Chassis

Mountain Computer

Extender Card

John Bell Engineering
SSM Microcomputer Products

Fan

MR Engineering
R.H. Electronics

Game I/O Expansion

CJM Industries
Programma

Hobbyist

John Bell Engineering
    Display Board
MR Engineering
    Computer Listener

IEEE-488 Interface

California Computer Systems
SSM Microcomputer Products

Industrial

United Detector Technology
    Optical Position Indicator

Isolated Parallel Power Interface

Interactive Structures

Joystick/Paddle

Stanley van Dusen
Programma International

Keyboard

RCA

Keyboard Enhancer

Dockside Computing
Lazer Micro Systems
Videx

Keypad

Advanced Business Technology

Light Pen

3-G Co.

Lower Case Adapter

Dan Paymar
Lazer Micro Systems
MUSE
Videx

Mobile Power Supply

CPU Inc.

Modem, Direct Connect

Hayes Microcomputer Products
Novation

Modem, Direct Couplers

Livermore Data Systems
Micromate Electronics

Modem, Acoustic Couplers

Racal-Vadic
U.S. Robotics

Multi-User System

Corvus Systems
Nestar Systems

Music Keyboard

Syntauri Inc.
Passport Designs

Music Synthesizer

ALF Products
Mountain Computer
Passport Designs

Optical Card Reader

Mountain Computer

Paper Tape Reader/Punch

GNT Automatic

Plotter

Houston Instruments
Mauro Engineering

Processor Card

Carl Dick, Distributor
    Z-80A (4MHz)
Microsoft Consumer Products
    Z-80
Stellation Two
    6809

Prototyping Card

SSM Microcomputing Products

RF Modulator

ATV Research
M&R Enterprises

Remote Home Control

John Bell Engineering
Intelligent Control Systems
Mountain Computer
SciTronics
Thunderware
Trillium Group
    (bi-directional)

ROM Programmer

MicroProducts Inc.
Mountain Computer

ROM Interface

Mountain Computer

ROM Simulator

Lamar Instruments

Selectric/Typewriter Interface

Escon Products
Rochester Data

Serial RS232 Interface

John Bell Engineering
California Computer Systems
Prometheus Products
SSM Microcomputer Products

Sound Effects

MR Engineering

Speech Synthesis/ Recognition

Mountain Computer
Street Electronics
Votrax

Video 80 Column Display

Advanced Logic Systems
Bit-3 Computers
Computer Stop
    Double Vision
M&R Enterprises
Spies Labrotories
    Double Hi-Res Card
Videx

Video Hi-Res Enhancer

Ray Dahlby Electronics

Video Image Digitizing

Micro Works

Video Monitor

Amdek Corp.
Micro Products Unltd.

Video Standardizer

Adwar Video

Video Tape Recorder Controller

BCD Assoc.
Cauri Systems

Video Terminal

Hazeltine Corp.
SOROC Technology

Voice Entry Terminal

Scott Instruments

SOFTWARE PRODUCTS

For an extremely comprehensive list of software for both the Apple II and the Apple ///, take a look at VanLove's Software Directory. Copies are available from:

Vital Information, Inc.
350 Union Station
Kansas City, MO 64108
Phone: 913-384-3860

Accounting/Payroll/Billing

Charles Mann & Assoc.
ESP Computer Resources
Peachtree Software
Software Technology
    for Computers
Spectrum Software
Tercer Medio
Westware Software

Amateur Radio

High Sierra Software

Assembler/Disassembler/ Linker/Editor/Monitor

Decision Systems
Eastern House Software
Hayden Book Co.
High Sierra Software
    Word Type
Image Computer Products
LJK Enterprises
Quality Software

Astronomy

Information Unlimited Software

Aviation/Flight Simulation

Illinois Computer Mart
subLogic

Business

BPI
Broderbund
    Payroll (Pascal)
Compumax
    Order Entry
    General Ledger
Computer Consulting
Continental Software
High Sierra Software
Information Technologies Inc.
Mini Microcomputer Software
Organic Software
Personal Software
    Desktop/Plan
    Visicalc
Programma International
Retail Sciences
    Peachtree Software
Software Technology
    for Computers

Database Manager

ESP Computer Resources
Hayden Book Co.
High Technology
Highlands Computer Services
Personal Software
Programma International
Software Publishing Corp.
    PFS:
Stoneware Microcomputer Products
    DB Master
    Micro Memo
United Software of America
Westware Software

Decision Aids

DecisionMaster
    DecisionMaster

Depreciation

Aardvark Software

Educational Software

Compu TA
Computer Services Company
Computer Systems Design Group
Edu-Comp, Inc.
High Sierra Software
Mastertype
Micro Learningware
Stoneware

Electronic Design/Control

High Sierra Software
Hayden Book Co.
Howard S. Sams & Co., Inc.
Spectrum Software

Estate Planning

Aardvark Software

Financial Planning/Forecasting

Charles Mann & Assoc.
Decision Master
Denver Software
Educational Programming
    Systems
ESP Computer Resources
Hayden Book Co.
DR Jarvis Computing
Personal Software
    Desktop/Plan II
Powersoft
Spectrum Software

Foreign Language Applications

Tercer Medio

Games/Entertainment

Adventure International
Avant-Garde Creations
Automated Simulations
Broderbund Software
Continental Software
Creative Computing Software
Crystal Computer
Edu-Ware
    The Prisoner
Galaxy
Hayden Book Co.
High Sierra Software
Highlands Computer Services
Instant Software
Powersoft
Programma
Quality Software
Rainbow Computing
Sirius Software
Spectrum Software
Strategic Software
Stoneware Microcomputer Products
Strategic Simulations
Synergistic Software
Systems Design Software
United Software of America
Western MicroData Enterprises

Graphics

Connecticut Information Systems
Datasoft
Micro Co-op Software
Sirius Software
Versa Computing
West Coast Consultants

Home Applications

Continental Software
High Sierra Software
Soft Touch
    Recipes
    Budget

Inventory

Softech International Corporation
Software Technology
    for Computers

Languages/Compilers/Aids

Abacus Software
    Tiny Pascal
Application Software Development
Applied Analytics
Computer Systems Design Group
Decision Systems
Hayden Book Co.
Microsoft Consumer Products
Programma
Prometheus Products

Mailing List

Avant-Garde Creations
Computer Services Company
Continental Software
Hayden Book Co.
Powersoft

Mathematics/Numerical Analysis

Action-Research Northwest
Interactive Microware
Spectrum Software
Systems Design Software

Medical/Dental

Charles Mann & Assoc.
CompuSoCo
Hayden Book Company
Health Data Products, Inc.
Med Logic Systems
Professional Medical Software
Prosoft

Multi-User/Multi-Tasking

Omega Research

Pascal Programs

Abacus Software
Arizona Computer Systems
Broderbund
    Payroll
Link Systems

Program Editor/Cross-Reference

Highlands Computer Services
    CRAE 2.0

Real Estate

Continental Software

Scheduling/Appointments/Calendar

Organic Software
Spectrum Software
Stoneware Microcomputer
    Products

Scientific Applications

High Sierra Software

Statistics

Spectrum Software

Stock/Quotes

RTR Software
Hayden Book Company

Tax Planning/Preparation

Aardvark Software

Telecommunications/Terminal Driver

Microcom
    Telex & TWX
Software Sorcery
Southeastern Software
Southwestern Data Systems
    ASCII Express
    Z-Term
Telephone Software Connection

Text Editor/Word Processor

Charles Mann & Assoc.
Hayden Book Co.
Information Unlimited Software
Muse Software
Peachtree Software
Personal Business Systems
Programma International
Select Information Systems
Sympathetic Software
Systems Design Software

Utilities/Diagnostics

Compu-Tron Software Services
    Memory Page Editor
High Sierra Software
Nikrom Technical Products
Rak-Ware
    DISASM
XPS Inc.

Utilities/Disk/File/General

Dakin5
Hayden Book Co.
Highlands Computer Services
    MCAT 2.0
Image Computer Products
Jasac+
Omega Software Systems
Progressive Software
Sensible Software
United Software of America
Zeus Computing, A1

Utilities, Pascal

Advanced Business Technology
Stellation Two
    Pascal Speed-up Kit

Utilities/ROM

Highlands Computer Services
    CRAE 2.0
    MCAT 2.0
Soft CTRL Systems
    Applesoft Editor
    Applesoft Renumber
    'FID'
    Disk copy
Program Line Editor

Printers & Printing Terminals

Anadex, Inc.
9825 De Soto Ave.
Chatsworth, CA 91311
(213) 998-8010

Anderson-Jacobson, Inc.
521 Charcot Ave.
San Jose, CA 95131
(408) 263-8520

Axiom Corp.
5932 San Fernando Rd.
Glendale, CA 91202
(213) 245-9244

Capitol Circuits
Printer Products Div.
24 Denby Rd.
Allston, MA 02134
(617) 787-2030

Printers & Printing Terminals continued

Centronics
Route 111
Hudson, NH 03051
(603) 883-0111

C. Itoh Electronics
5301 Beethoven St.
Los Angeles, CA 90066
(213) 390-7778

DataSouth Computer Corp.
4740 Dwight Evans Rd.
Charlotte, NC 28210
(704) 523-8500

Diablo

Epson America Inc.
23844 Hawthorne Blvd.
Torrance, CA 90505
(213) 378-2220

Howard Industries Inc.
2031 E. Cerritos Ave., Bldg. 7K
Anaheim, CA 92806
(714) 778-3443

Integral Data Systems
Milford, NH 03055
(603) 673-9100

InterSell
465 Fairchild Dr., #214
Mountain View, CA 94043
(415) 964-5460

Malibu
8900 Eton Ave.
Suite G
Canoga Park, CA 91304
(213) 998-7694

Micro Peripherals Inc.
4426 S. Century Dr.
Salt Lake City, UT 84107
(801) 263-3081

NEC Information Systems Inc.
5 Militia Dr.
Lexington, MA 02173
(617) 862-3120

Okidata Printers/Eakins Assoc. Inc.
999 Independence Ave.
Mountain View, CA 94043
(415) 969-4533

Pertec
12910 Culver Blvd.
Los Angeles, CA 90066
(213) 822-9222

Printronix, Inc.
17421 Derian Ave.
Irvine, CA 92714
(714) 549-8272

Qantex
60 Plant Ave.
Hauppauge, NY 11787
(516) 582-6060

Qume Corp.
2350 Qume Dr.
P.O. Box 50039
San Jose, CA 95150
(408) 942-4200

Trendcom
484 Oakmead Pkwy.
Sunnyvale, CA 94086
(408) 737-0747

User Groups

APPLE USERS

WORLD WIDE

APPLE USERS GROUP SWEDEN
Johan Nilson
Norra Vallvagen 24
Kristianstad, 291 32 Sweden
Phone: (UNAVAILABLE)

APPLE USERS CLUB
Peter Kazacos-Treasurer
8 Leemon Street
Condell Park-NSW, Australia 2200
Phone: (UNAVAILABLE)

APPLE TURNOVER
Karl Holmes, Barbara Central School
West Street
Barbara-NSW, Australia 2347
Phone: (UNAVAILABLE)

TAS APPLE USERS CLUB
Georgia Cook-Secretary
422 Elizabeth Street
North Hobart
Tasmania, Australia 7000
Phone: 349616

APPLE USERS CLUB WESTERN AUSTRALIA
Tim Russell-President
269 Marmion Street
Cottlesloe-W. A., Australia 6011
Phone: 09-457-1555

MICOM
Stephen Dart-President
P. O. Box 60
Canterbury-Victoria, Australia 3126
Phone: 03-509-9729

W. A. U. G.
Dr. Patrick Lip-President
P. O. Box 19
Wondai, Qld., Australia 4606
Phone: 074-9922-84

APPLE USER CLUB AUSTRIA
Werner Beyerle-President
P. O. Box 51
A-1181 Wien, Austria
Phone: 0043-222-652795

APPLE B.C. USERS SOCIETY
Gary Little-President
#101-2044 West Third Avenue
Vancouver B. C. Canada
Phone: 604-731-7886

BRAZIL APPLE CLUBE
Dr. Luiz E. Pellanda-President
Rua Maestro Pena, 90
Porto Alegre, Brazil 90 000
Phone: 0512-23-0577

APPLE—CAN
Louis H. Milrad-President
P. O. Box 696 Station B
Toronto-Ontario, Canada
M2K 2P9
Phone: 416-223-0599

CLUB APPLE DE MONTREAL
Michel Cartier-President
10,265 Hamelin
Montreal, Canada H2B 2E7
Phone: (UNAVAILABLE)

OTTAWA 6502 USERS GROUP
Paul Irwin-President
P. O. Box 6283 Station J
Ottawa-Ontario, Canada K2A 1T3
Phone: 613-728-6728

CLUB DE MICRO-ORDINATEUR ST-JEAN
Ronald Leger-Secretary
P. O. Box 21
St. Jean-Quebec, Canada
J3B 6Z1
Phone: (UNAVAILABLE)

BOLO/UB APPLE CLUB
Pierre Beaudin-President
1208 Patenaude #3
Laval-Quebec, Canada H76 3H2
Phone: 514-663-2771

GRUPO USARIOS APPLE DE COLUMBIA
Jorge Ladron de Guevara
A. A. 91226
Bogota, Columbia
Phone: 249-71-85

BRITISH APPLE SYSTEMS USER GROUP
John Sharp-Chairman
P. O. Box 174
Watford, England WD2 6NF
Phone: 09273-75093

OEDIP—APPLE
Laurent de Vilmorin
8 Place Ste Opportune
Paris, France 75001
Phone: 1-5084621

APPLE CLUB ROEDINGHAUSEN
Axel Vogt-President
Wehmerhorstsr. 110
Roedinghausen, Germany
D-4986
Phone: (UNAVAILABLE)

HONG KONG APPLE
C/O Delta Communication
15 Cumberland Road Rear Portion
Kowloon Tong, Hong Kong
Phone: (UNAVAILABLE)

HONG KONG APPLE DRAGON
Robert Gliss-Stephens
Finance Ltd.
101 Fu House
7 Ice House Street, Hong Kong
Phone: 5-210295-8

APPLE USERS—DUBLIN
Simon Stewart-President
51 Lower Camden Street
Dublin 2, Ireland
Phone: 751484

YEDA—NIKUV COMPUTERS
Moshe Bornovski-President
12 Karlibach Street
Tel-Aviv, Israel
Phone: 03-2259-151

BAKED APPLE—K. YAMADA
Miyadaira Apts No. 1; 1575 Sugao
Takatsu-Ku
Kawasaki-Shi, Kangawa Japan
Phone: (UNAVAILABLE)

APPLE ORCHARD
Tony Lee; 23A; Jalan Jejaka
Tujuh
Taman Maluri; Batu 3; Jalan
Cheras
Kuala Lumpur, Malaysia
Phone: (UNAVAILABLE)

APPLE GEBRUIKERS GROEP
NEDERLAND
P/A J. P. Haas-Secretary
Bergselaan 145A
Rotterdam, Netherlands
Phone: (UNAVAILABLE)

NZ GROUP OF APPLE USERS
T. Stallknecht-President
90 Washington Avenue
Brooklyn 2, New Zealand
Phone: 894800

EMU
Ian Webster-Coordinator
Box 3143; G. P. O.
Sydney 2001, NSW Australia
Phone: (UNAVAILABLE)

APPLE BUGS
Toshikazu Yamashita
20-29 Banchi Yamata-Cho 1
Chome
Suita City, Osaka Japan
Phone: (UNAVAILABLE)

APPLE P. I.
Benjamin S. Jalandoni
3rd Floor/Liberty Bldg/Pasay
Road
Makati/Manila, Philippines 3116
Phone: 88-70-36

APPLE—EDEN
Colin O'Hara-President
10 Seton Terrace
Glasgow, Scotland G31 2HU
Phone: 041-554-3664

S AUSTRALIAN APPLE USERS
CLUB
Terry Neal
C/O Computerland
125 Pirie Street
Adelaide, S. Australia 5000
Phone: 08-223-5808

TAC2 APPLE USERS GROUP
Harry Brindley-Secretary
P. O. Box 87421
Houghton, South Africa 2041
Phone: (UNAVAILABLE)

CATALUNYA APPLE CLUB
Sr Rife Uriol-President
Fabra Y Puig 389 E/4
Barcelona-31, SPAIN
Phone: 254-7909

APPLE CLUB ZAGREB
Zeljko Lalic-Dipl. Zing
Ruzmarinka 3/II
41000 Zagreb, Yugoslavia
Phone: (UNAVAILABLE)

APPLE USERS OF PARAGUANA
Gary Menszyk
G. Sanderson
C/O Lagoven S. A. Apt. 47
Judibana; Faclon, Venezuela
4147A
Phone: 58-69-51601

APPLE OF EUROPE
Wolfgang Dederichs-President
P. O. Box 4068
Hattingen, W. Germany D-4320
Phone: 02324/67472

APPLE CLUB FRANKFURT
Lothar Rochstroh-President
Schweizer Str. 92
Frankfurt/M. 70, W. Germany
D-6000
Phone: (0611)-61-45-12

UNITED STATES

Alabama

APPLE CORPS OF BIRMINGHAM
Bob Feezor-President
2931 Pahokee Trace
Birmingham, AL 35243
Phone: 205-967-4261

QUAD CITIES APPLE BYTERS
Peter A. Eckhoff-President
129 E. Oak Hill Drive
Florence, AL 35630
Phone: (UNAVAILABLE)

NEWTON'S TREE APPLE USER GROUP
Frank H. Emens-President
3714 Lakewood Circle
Huntsville, AL 35811
Phone: 205-852-0537

Arkansas

LITTLE ROCK APPLE ADDICTS
Chris Johnson-Treasurer
P. O. Box 55215 Hillcrest Station
Little Rock, AR 72205
Phone: 501-568-5059

Arizona

ADAM II
Alan Sawyer-President
P. O. Box 34056
Pheonix, AZ 85206
Phone: 602-248-4595

GILA VALLEY APPLE GROWERS ASSN
Don Lancaster-C/O Synergetics
P. O. Box 1077
Thatcher, AZ 85552
Phone: 602-428-4073

MOUNTAIN VIEW APPLE USERS GROUP
Joseph J. Cracchiolo-President
1923 Viola Drive
Sierra Vista, AZ 85635
Phone: 602-458-2332

TUCSON APPLE USERS GROUP
Dan Davidson-LCS
Pima College-2202 W. Anklam Road
Tucson, AZ 85709
Phone: 602-884-6000

California

APPLE FOR THE TEACHER
Ted Perry-President
5848 Riddio Street
Citrus Heights, CA 95610
Phone: (UNAVAILABLE)

APPLE SAC
Bill Norris-President
8074 Ruthwood Way
Orangevale, CA 95662
Phone: 916-381-4166

SAN FRANCISCO APPLE CORE
Randy Fields-President
1515 Sloat Blvd.-Suite #2
San Francisco, CA 94132
Phone: 415-556-2342

ABACUS USER GRP
Ed Avelar-President
2850 Jennifer Drive
Castro Valley, CA 94546
Phone: 415-538-2431

MIDWAY COMPUTER CLUB
John Yantis-President
506 Ridgewood Drive
Vacaville, CA 95688
Phone: 707-448-8430

SILICON APPLE PROGRAMMERS SOCIETY
Lowell Noble-President
18138 Bancroft Avenue
Monte Sereno, CA 95030
Phone: 408-354-6120

L. A. APPLE USERS GROUP
Philip Wasson-President
9513 Hindry Place
Los Angeles, CA 90045
Phone: 213-649-1428

SANTA CRUZ APPLE GROUP
Jim McCaig-President
P. O. Box 1428
Santa Cruz, CA 95061
Phone: 408-335-8750

APPLE P.I.E.
Bill Nienhaus-President
337 Montclair
Santa Clara, CA 95051
Phone: 408-247-6470

S. P. A. C. E.
Bob Davies-President,
Computerland
4546 El Camino Real
Los Altos, CA 94022
Phone: 415-493-8330

APPLE MUG
C/O Med Logic Systems
2030 East 4th Street #133
Santa Ana, CA 92705
Phone: 714-953-9151

THE 'PITS' OF SANTA BARBARA
Don Cobb-President
3835 Connie Way
Santa Barbara, CA 93101
Phone: 815-969-5607

APPLE PEELERS
Gene Wilson-President
340 N. Civic Drive Apt. #503
Walnut Creek, CA 94596
Phone: 415-878-0789

ORIGINAL APPLE CORPS
Kip Reiner-President
12804 Magnolia
Chino, CA 91710
Phone: (UNAVAILABLE)

HESEA APPLE COMPUTER CLUB
Bud Grove-President
21111 Dolores #146
Carson, CA 90745
Phone: 213-549-9664

APPLE CREEK
Henry Couden, Computerland
1815 Ygnacio Valley Road
Walnut Creek, CA 94598
Phone: 935-6502

LERC ACES
Alan G. Hyde-Secretary
P. O. Box 551
Burbank, CA 91520
Phone: 213-899-2323

APPLE PI
C/O Computerland-Marion Clarke
171 E. Thousand Oaks Blvd.
Suite 104
Thousand Oaks, CA 91360
Phone: 805-495-3554

JPL COMPUTER CLUB / JPL APPLE CLUB
Eliot Goldyn-President
24575 Spartan Street
Mission Viejo, CA 92691
Phone: 213-354-7009

APPLE BUG
Gary Atchinson-President
4509 Millbrook Way
Bakersfield, CA 93309
Phone: 805-831-7723

EVAC
Timothy Malone-Treasurer
250½ W. Center Apt. B
Covina, CA 91723
Phone: 714-332-7690

UCLA APPLE USERS GROUP
Philip B. Ender-President
17565 Bullock Street
Encino, CA 91316
Phone: 213-825-1944

APPLE-CORPS OF SAN DIEGO
Dennis A. Rosemier-Treasurer
279 Satinwood Way
San Diego, CA 92114
Phone: 714-479-6512

APPLE JACKS
Al Johnson-President
4818 Reese Road
Torrance, CA 90505
Phone: (UNAVAILABLE)

APPLE/VALLEY COMPUTER CLUB
John Stankiewiz-President
4900 Newcastle
Encino, CA 91316
Phone: 213-345-8507

APPLEHOLICS ANONYMOUS
Don Wilson-C/O Byte Shop
155 Morse Avenue
Ventura, CA 93003
Phone: 805-647-8945

AEROSPACE APPLE USER GROUP
Dwight U. Phillips-President
28901 Lotusgarden Drive
Canyon Country, CA 91351
Phone: 805-251-1516

SOUTH BAY APPLES COMPUTER CLUB
Frank Jedziniak-President
P.O. Box 5201
Torrance, CA 90510
Phone: 213-539-1200

APPLEPICKERS
Steve Wilder-President
P. O. Box 4208
Santa Rosa, CA 95402
Phone: 707-544-4783

HFEA APPLE COMPUTER USERS GROUP
Don Andert-President
417 Meadowbrook Place
Anaheim, CA 92801
Phone: 714-776-6384

TRI-NETWORK APPLE USERS GROUP
Jeff Mazur-President
8041 Sadring
Canoga Park, CA 91304
Phone: 992-4993

RIDGECREST APPLE GROUP
Gene Thomas-President
Star Route Box 109E
Inyokern, CA 93527
Phone: (UNAVAILABLE)

SOURCE APPLE USERS GROUP
Joel L. Amronin-Treasurer
2525 Beverly Avenue #9
Santa Monica, CA 90405
Phone: 213-3396-8668

BLOSSOM VALLEY APPLE CLUB
Frank E. Brinkman-President
5821 Cottle Road
San Jose, CA 95123
Phone: 408-578-2815

NORTH COUNTY COMPUTER CLUB
Jim Frazee-Frazee Inc.
2521 Oceanside Blvd.
Oceanside, CA 92054
Phone: 714-433-6021

S. M. A. L. APPLE
Charles Baca-President
223 S. Broadway
Santa Maria, CA 93454
Phone: 805-925-6675

Colorado

APPLE PI USERS GROUP
Scott Knaster-President
P. O. Box 17467
Denver, CO 80217
Phone: 303-355-2379

SOUTH COLORADO APPLE USERS
Tom Thomas-President
1635 S. Prairie
Pueblo, CO 81005
Phone: 303-564-3545

Connecticut

APPLE USERS OF WESTPORT
Jack Adinolfi-President
Computerworks
1439 Post Road East
Westport, CT 06880
Phone: 203-227-6854

APPLELIST
Joseph Cohen-Secretary
50 Ida Lane
West Haven, CT 06516
Phone: 203-397-1407

NEW LONDON APPLE USERS GROUP
Ronald Gibson, Computer Lab
130 Jefferson Avenue
New London, CT 06320
Phone: 203-447-1079

Washington D. C.

WASHINGTON APPLE PI
John Moon-President
P. O. Box 34511
Washington, D.C. 20034
Phone: 202-332-9102

Delaware

GRAPE
Frank Weinberg-President
P. O. Box 8904
Newark, DE 19711
Phone: 302-738-6365

Florida

A. C. E. S.
Don Lehmbeck-President
P. O. Box 9222
Coral Springs, FL 33065
Phone: 305-941-7252

MAUG
Steve Pierce-President
2300 N. W. 135 Street
Miami, FL 33167
Phone: 305-595-8728

SCAT
Sandy Bernstein-President
P. O. Box 7488
Clearwater, FL 33518
Phone: 813-961-5705

APPLE USERS CORE
Pete Seals
307 Tarpon Road
Mary Esther, FL 32569
Phone: 581-0002

SUN COAST COMPUTER ASSN
Jim Parrish-President
P.O. Box 15294 Southgate PO
Sarasota, FL 33579
Phone: 813-485-2564

APPLE JAX
Ed Dunn-President
1021 King Street
Jacksonville, FL 32204
Phone: (UNAVAILABLE)

SPACE COAST APPLE USER GROUP
Billy M. Washam-President
P. O. Box 4332
Patrick AFB, FL 32925
Phone: (UNABAILABLE)

APPLE PI OF BREVARD
Tony R. Marshall-President
P. O. Box 327
Melbourne, FL 32901
Phone: 305-725-4328

SMAUG
Phil Mitchell-President
10201 Fontainebleau Blvd. #206
Miami, FL 33172
Phone: 305-551-1000

Georgia

ATLANTA SOCIETY OF PROF APPLE USERS
Jerry Long-President
6600 Powers Ferry Road
Suite 220
Atlanta, GA 30339
Phone: 404-955-2663

SEA
Andy Morton-President
3258 Powers Ferry Road
Marietta, GA 30067
Phone: 404-977-8600

Hawaii

H. A. U. S.
Bob McDowell-President
P. O. Box 30262
Honolulu, HI 96820
Phone: 808-261-3733

Iowa

THE GREEN APPLES
Tom Jacobsen-President
4417 N. Zircon Lane Lot 129
Cedar Falls, IA 50613
Phone: 319-268-0572

GLITCH KICKERS COMPUTER CLUB
Joe Brazzle-Treasurer
3711 Douglas
Des Moines, IA 50310
Phone: 515-265-6266

IOWA CITY APPLE USERS GROUP
Dave Thomas-President
134 Ravencrest Drive
Iowa City, IA 52240
Phone: 319-353-3170

CONDUIT
J. W. Johnson-100 Lindquist Center
Burlington & Madison Streets
Iowa City, IA 52242
Phone: (UNAVAILABLE)

CEDAR RAPIDS APPLE USERS GROUP
Pete Tillman-President
417 Third Avenue
Cedar Rapids, IA 52404
Phone: 319-366-6327

I/OWA USER GROUP
Joann Short-Secretary
844 10th N.E.
Mason City, IA 50401
Phone (UNAVAILABLE)

AGRI-CURSORS
Neil Stadlman-C/O Sac City State Bank
500 Audobon
Sac City, IA 50583
Phone: 712-622-4721

Idaho

A. B. U. G.
Michael Coombs-Secretary
1505 Ressigue
Boise, ID 83702
Phone: 208-345-7149

P. I. N. E. APPLES
Hugh Tucker-Secretary
1855 Jean Street
Pocatello, ID 83201
Phone: (UNAVAILABLE)

Illinois

NW SUBURBAN APPLE USERS
Michael L. Robins-Vice President
1300 S. Elmhurst Road
Mt. Prospect, IL 60056
Phone: 312-593-2709

C. A. C. H. E.
Tim Clark-President
18W 145 Bel Air Court
Darien, IL 60559
Phone: 312-447-6267

APPLE PI COMPUTER CLUB
Jack Gratz-President
11630 S. Nagle Avenue
Worth, IL 60482
Phone: 312-448-6548

DUPAGE APPLE USER'S GROUP
Glenn Young-President
10 S. 592 Windjammer
Naperville, IL 60540
Phone: 312-420-8505

COMSAT
Kile Mullen-President
12 Crossroads Plaza
Granite City, IL 62040
Phone: 618-452-1860

D. A. T. A.
John Pausteck-Treasurer
5048 Pebble Creek Trail
Loves Park, IL 61111
Phone: 815-633-1569

SLACC
Dennis W. Jolly-President
2445 Cleveland
Granite City, IL 62040
Phone: 618-451-6502

CIA (Central Illinois Apple)
Gary Benway-President
1023 W. Hudson
Peoria, IL 61604
Phone: 309-444-9705

CRAB-APPLES
Robert F. Gonsowski-President
P. O. Box 437
Desoto, IL 62924
Phone: (UNAVAILABLE)

Indiana

THE APPLE PICKERS INC.
Jerry Baker-President
1620 E. 83rd Street
Indianapolis, IN 46240
Phone: 317-251-5181

CAUG
Charles Hatcher-President
2805 Chestnut Court
Columbus, IN 47201
Phone: (UNAVAILABLE)

APPLE TECH
Michael A. Miller-President
412 West Third
Mishawaka, IN 46544
Phone: (UNAVAILABLE)

FORT WAYNE APPLE COMPUTER USER GROUP
Joe Kucharski-President
3833 Foresthill Avenue
Fort Wayne, IN 46805
Phone: 219-485-3388

Kansas

APPLEBUTTER
Michael Frame-President
10049 Santa Fe Drive
Overland Parks, KS 66212
Phone: 913-884-8529

APPLE BITS
Robert K. Mills
6140 Glenwood
Mission, KS 66202
Phone: 913-236-8679

PLANE APPLE CLUB
John Van Walleghen-President
Box 12013
Wichita, KS 67277
Phone: 316-522-8410

Kentucky

L. A. U. G. H. S.
Dr. Michael Finn
8207 Pipilo
Louisville, KY 40222
Phone: 502-426-3815

Louisianna

B. R. A. N. C. H.
John Rupp-President
324 W. Parker Blvd. #35
Baton Rouge, LA 70808
Phone: 504-766-62265

CRESCENT CITY APPLE CORE
John Downing-President
72 Old Hickory Avenue
Chalmette, LA 70043
Phone: 504-246-8438

CENLA APPLE
Russel A. Pauley-President
Box 1564
England AFB, LA 71301
Phone: (UNAVAILABLE)

Massachusetts

N. E. A. T.
Lori Steinmetz-President
P. O. Box 2652
Woburn, MA 02155
Phone: 617-767-1722

APPLESEED
Don Isaac-President
17 Saxon Road
Worcester, MA 01602
Phone: 607-755-2126

APPLE CORE OF BERKSHIRE COUNTY
Scott Rodman-President
32 Deborah Avenue
Pittsfield, MA 01201
Phone: 413-442-4759

APPLE/BOSTON
Boston Computer Society
3 Center Plaza
Boston, MA 02108
Phone: 617-367-8080

APPLESAUCE
Leon A. Osborne-President
118 Brookhaven Drive
East Longmeadow, MA 01028
Phone: (UNAVAILABLE)

Maryland

MARYLAND APPLE CORPS
Joe Lewis-President
C/O Computers Etc.
13A Allegheny Avenue
Towson, MD 21204
Phone: 301-256-3560

PENCOM
John A. Pence-President
J-303 Waverly Drive
Frederick, MD 21701
Phone: 301-662-1997

Michigan

MICHIGAN APPLE COMPUTER CLUB
Jim Niemann-President
P. O. Box 551
Madison Heights, MI 48071
Phone: 313-353-7648

K. A. C. U. S.
Gary Wilkins
Computer Room
455 W. Michigan Avenue
Kalamazoo, MI 49007
Phone: 616-381-6476

GRAND RAPIDS APPLE
Tim Hartley-Secretary
3268 Coach Lane #2A
Kentwood, MI 49508
Phone: (UNAVAILABLE)

ANN ARBOR APPLE
Terry Wynn-President
P. O. Box M-1047
Ann Arbor, MI 48106
Phone: (UNAVAILABLE)

APPLE CORE EXAMINERS
Mark Turmell-President
4691 S. Elm Drive
Bay City, MI 48706
Phone: 517-684-9189

Minnesota

MINI'APP'LES
Daniel Buchler-President
13516 Grand Avenue South
Burnsville, MN 55337
Phone: 612-890-5051

Missouri

MICRO/PERSONAL COMPUTER CLUB
Rick Connolly-President
41 Roland Drive
Ballwin, MO 63011
Phone: 314-227-6702

APPLE JACKS
Fred Bruner-Secretary
11145 Suntree Road, Apt. D
St. Louis, MO 63138
Phone: 314-869-9050

APPLE SQUIRES OF THE OZARKS
C/O Milton Rhoads-President
1904 E. Meadowmere
Springfield, MO 65804
Phone: 417-862-6500

APPLE EYE
Michael B. Rumeld M.D.-President
25 Morwood Lane
Creve Coeur, MO 63141
Phone: 314-569-2762

A. M. M. P. L. E.
Allen Hahn-President
333 E. Winter
Columbia, MO 65201
Phone: 314-443-0689

Nebraska

COMPUSERS
Dorothy G. Friend-Secretary
P. O. Box 2064
Hastings, NB 68901
Phone: (UNAVAILABLE)

North Carolina

CAROLINA APPLE CORE
Gene Jackson-Vice President
P. O. Box 31424
Raleigh, NC 27622
Phone: 919-781-3755

GREEN APPLES
Nancy Terrell-President
218 N. Elm Street
Greensboro, NC 27401
Phone: 919-275-2983

APP-LE-KATIONS
Lew Scott
6525 Springfield Drive
Charlotte, NC 28212
Phone: 704-554-8709

North Dakota

G.F. APPLE S. A. U. C. E.
Paul Kobe-C/O Computerland
2500 B South Columbia Road
Grand Forks, ND 58201
Phone: 701-746-0491

APPLE POLISHERS
Craig Nansen-President
1112 Glacial Drive
Minot, ND 58701
Phone: 701-838-6444

Nebraska

APPLESAUCE OF OMAHA
Bill Judd-President
Computers West
7435 Pacific Street
Omaha, NE 68124
Phone: 402-391-3737

New Jersey

APPLE GROUP-NJ
Steve Toth-President
1411 Greenwood Drive
Piscataway, NJ 08854
Phone: 201-968-7498

SOUTHERN NJ APPLE USERS GROUP
Larry Margulis-President
106 Ashbrook Road
Cherry Hill, NJ 08034
Phone: 609-428-4429

PRINCETON APPLE USERS GROUP
Hans Jorgensen-President
C/O Computer Encounter
2 Nassau Street
Princeton, NJ 08540
Phone: (UNAVAILABLE)

MONMOUTH APPLE CORPS
David C. MacMakin-President
P. O. Box 333
West Long Beach, NJ 07764
Phone: 201-870-9453

SHORT HILLS APPLE PITS
Steve Bloch-President
29 Clive Hills Road
Short Hills, NJ 07078
Phone: 201-376-8966

New Mexico

APPLEQUERQUE COMPUTER CLUB
James Segrest-President
6609 Orphelia Avenue NE
Albuquerque, NM 87109
Phone: 505-821-7418

MESILLA VALLEY ORCHARD
John Mitchner-President
P.O. Box 114
Las Cruces, NM 88001
Phone: 505-526-4218

Nevada

APPLE CORPS OF S. NEVADA
Sandy Tiedeman-President
6325 Portola Road
Las Vegas, NV 89108
Phone: 702-647-6502

New York

SUFFOLK APPLE COMPUTER SOCIETY
M. Weinstock-President
64 Pinedale Road
Hauppauge, NY 11787
Phone: 516-360-0988

BIG APPLE USERS GROUP
Tony Cerreta-President
P.O. Box 490
Bowling Green Station
New York, NY 10274
Phone: 914-636-3417

APPLE POWER
Jim Lyons-President
21 Ridgedale Avenue
Farmingville, NY 11738
Phone: 516-248-8080

U. A. U. G. C/O UPSTATE COM.
Bill Etter-President
629 French Road
New Hartford, NY 13413
Phone: 315-399-1139

MID HUDSON MICRO USERS
William Quimby Jr.-President
Imperial Plaza
Wappingers Falls, NY 12590
Phone: 914-297-1223

TSAUG APPLE CLUB
Tom Coulter-President
216 Cherry Road
Syracuse, NY 13219
Phone: 315-468-4262

APPLE C. I. D. E. R.
C/O Jim Berube-President
1435 Tudor Way
Victor, NY 14564
Phone: 716-924-7705

CAMS—APPLE USERS GROUP
Stanley L. Mathes-President
Box 348 Ridge Road-R.D. #1
Scotia, NY 12302
Phone: (UNAVAILABLE)

SUNY AT BUFFALO
S. D. Farr-Quantitative Analysis Lab
217 Baldy Hall
Amherst, NY 14260
Phone: 716-636-2110

Ohio

APPLE-DAYTON
Dick Peschke-Secretary
4819 Leafburrow Drive
Dayton, OH 45424
Phone: (UNAVAILABLE)

NEO-APPLE CORE
Tom Wysocki
7047 East Jefferson Drive
Mentor, OH 44060
Phone: 216-942-7086

APPLE-SIDERS
John Anderson-President
5707 Chesapeake Way
Fairfield, OH 45014
Phone: (UNAVAILABLE)

CENTRAL OHIO APPLE COMPUTER HOBBY
Thomas Mimlitch-President
1357 Bernard Road
Columbus, OH 43227
Phone: 237-3380

TOLEDO APPLE USERS
Larry J. Lewandowski-President
1417 Bernath Parkway
Toledo, OH 43615
Phone: 419-476-8463

Oklahoma

TULSA COMPUTER SOC-APPLE USERS
J. L. Shanks-President
P. O. Box 1133
Tulsa, OK 74101
Phone: 918-835-3926

OKC APPLE USERS GROUP
David A. Simpson-President
3600 N. W. 39th
Oklahoma City, OK 73112
Phone: 405-755-1260

MIDWEST CITY HOSPITAL APPLE USERS
Dr. W. E. McGuire
2825 Parklawn Drive
Midwest City, OK 73110
Phone: 405-732-6682

Oregon

CORVALLIS APPLE CLUB
Jack Trowbridge-President
2013 N. W. Monroe
Corvallis, OR 97330
Phone: 503-757-7496

A. P. P. L. E. PORTLAND
Will Newman II-Secretary
1915 N. E. Couch
Portland, OR 97232
Phone: 503-283-8361

Pennsylvania

APPLE USER OF PA
Neil Lipson-President
29 S. New Ardmore Avenue
Broomall, PA 19008
Phone: 215-356-6183

KEYSTONE APPLE CORE
David W. Murdoch
4644 Carlisle Pike
Mechanicsburg, PA 17055
Phone: (UNAVAILABLE)

ARG
Bill Hindorff-President
16 Laurel Lane
Glen Riddle, PA 19037
Phone: (UNAVAILABLE)

South Carolina

SCAPPLE
Felix Clayton-President
1610 Longview Road
Mt. Pleasant, SC 29464
Phone: 803-554-9171

AUGUSTA APPLE USERS GROUP
David Anderson-President
819 Jackson Avenue
N. Augusta, SC 29841
Phone: 803-279-4974

RAPID CITY APPLE USERS
James P. Gayton-President
3016 Glenwood
Rapid City, SD 57701
Phone: 605-343-2949

Tennesee

APPLE CORPS OF MEMPHIS
Steve Romeo-President
627 S. Mendenhall
Memphis, TN 38117
Phone: 901-761-4743

MUSIC CITY APPLE CORE
Rod Wagner-President
765 McMurray Drive Apt. #4
Nashville, TN 37211
Phone: 615-331-2287

Texas

APPLE CORPS
Bob Sander-Cederlof
P.O. Box 5537
Richardson, TX 75080
Phone: 214-324-2050

APPLESEED
Bill Hyde-President
P. O. Box 12455
San Antonio, TX 78212
Phone: 512-737-0213

HAAUGG
Bruce Barber-President
11803 Rowood Drive
Houston, TX 77070
Phone: 713-469-5805

MICRO APPLE CORE
Tom Munroe-President
3920 Caruth Blvd
Dallas, TX 75225
Phone: 691-6140

RIVER CITY APPLE CORPS
Curt Wyman-President
12404 Split Rail Parkway
Austin, TX 78750
Phone: 512-258-5486

A. I. D. E.
John Jackson-Editor
5700 Dixon
Amarillo, TX 79109
Phone: 806-352-3563

HOBBY COMPUTER INFORMATION EXCHANGE
Michael Green-President
6718 Spring Haven
San Antonio, TX 78249
Phone: 512-699-0146

FRANKLIN MOUNTAIN APPLE ORCHARD
Bill Evans
Drawer G
El Paso, TX 79951
Phone: 915-877-2383

APPLE PI OF THE PERMIAN BASIN
Clay Francell
415-E. 43rd. Street
Odessa, TX 79762
Phone: 915-333-3430

H.O.T.-APPLE-P. I. E.
Andrew Marquart-President
2321 Lee Street
Waco, TX 76711
Phone: 817-756-1690

ABILENE APPLE CLUB
Scott Byod-President
925 N. Judge Ely Blvd.
Abilene, TX 79601
Phone: 915-673-6708

FORT WORTH APPLE USER GROUP
Lee Meador
1401 Hillcrest Drive
Arlington, TX 76010
Phone: 817-461-1981

Utah

APPLE SLICE
Gary Allen-President
P. O. Box 536
Bountiful, UT 84010
Phone: 292-4555

Virginia

PENNINSULA APPLE CORE
Jerry Aycock-Secretary
1419 Todds Lane
Hampton, VA 23666
Phone: 804-827-0041

APPLE T. A. R. T.
Gordon Andrews-President
1706 Hanover Avenue
Richmond, VA 23220
Phone: 804-320-2260

APPLE WORMS
Bill Holmes-President
3307 Indigo Road
Virginia Beach, VA 23456
Phone: (UNAVAILABLE)

NOVAPPLE
Tom Lucas-Secretary
8108 Adair Lane
Sprintfield, VA 22151
Phone: 703-321-9593

Washington

A.P.P.L.E.—WASHINGTON
Dick Hubert-President
14109 S. E. 168th Street
Renton, WA 98055
Phone: 206-255-7410

KITSAP APPLE USERS GROUP
Darrell Dunmire-President
P. O. Box 1194
Silverdale, WA 98383
Phone: (UNAVAILABLE)

FETCH
Charles E. Cook
Naval Air Facility Box 13
FPO Seattle, WA 98767
Phone: 046-251-1520
Ext 228-6859

THE G.R.A.P.E.
Steve Lawson
P. O. Box 283
Port Orchard, WA 98366
Phone: 206-876-1397

AU
Chuck Vyverberg-President
12816 E. Desmet
Spokane, WA 99216
Phone: 509-489-2861

Wisconsin

ADAM & EVE APPLE GROUP
Mike Dhuey-President
11 S. Hancock Street
Madison, WI 53703
Phone: 608-256-5306

WISCONSIN APPLE USERS
C/O Cybernetic Mechanism
P. O. Box 11463
Milwaukee, WI 53211
Phone: 414-964-6645

Wyoming

THE APPLE NET
James E. Hassler
129 Park Ave.-Orchard Valley
Cheyenne, WY 82001
Phone: 307-632-4934

apple computer inc.

10260 Bandley Drive
Cupertino, California 95014
(408) 996-1010

1394